5.0 REAL MODE ARCHITECTURE

5.1 Introduction

When the Military Intel486 processor is reset or powered up, it is initialized in Real Mode. Real Mode has the same base architecture as the 8086 processor, except that it allows access to the 32-bit register set of the Military Intel486 processor. The Military Intel486 processor addressing mechanism, memory size and interrupt handling are identical to those of Real Mode on the 80286 processor.

All of the Military Intel486 processor instructions are available in Real Mode (except those instructions listed in section 6.5.4, ''Protection and I/O Permission Bitmap''). The default operand size in Real Mode is 16 bits, as in the 8086 processor. In order to use the 32-bit registers and addressing modes, override prefixes must be used. Also, the segment size on the Military Intel486 processor in Real Mode is 64 Kbytes, forcing 32-bit effective addresses to have a value less than 0000FFFFH. The primary purpose of Real Mode is to enable Protected Mode Operation.

The LOCK prefix on the Military Intel486 processor, even in Real Mode, is more restrictive than on the 80286 processor. This is due to the addition of paging on the Military Intel486 processor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The Military Intel486 processor can not require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore, the LOCK prefix can not be supported during repeated string instructions.

Table 5-1 lists the only instruction forms where the LOCK prefix is legal on the Military Intel486 processor.

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Because, on the Military Intel486 processor, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the Military Intel486 processor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

Table 5-1. Instruction Forms Where LOCK Prefix Is Legal

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
CHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem
CMPXCHG, XADD	Mem, Reg

5.2 Memory Addressing

In Real Mode the maximum memory size is limited to 1 megabyte. (See Figure 5-1.) Thus, only address lines A2–A19 are active. (Exception, after RESET address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed. See section 9.5, ''Reset and Initialization''.)

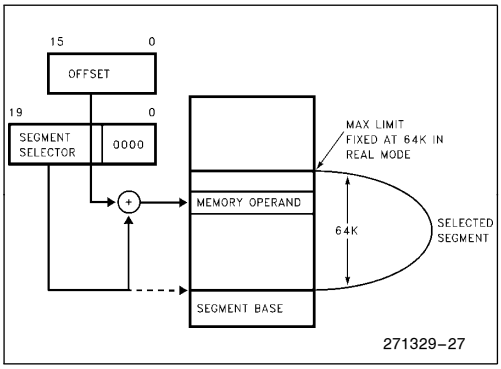


Figure 5-1. Real Address Mode Addressing

Because paging is not allowed in Real Mode, the linear addresses are the same as the physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register, which is shifted left by four bits to an effective address. This addition results in a physi-

cal address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Because segment registers are shifted left by 4 bits, Real Mode segments always start on 16-byte boundaries.

All segments in Real Mode are exactly 64-Kbytes long, and may be read, written, or executed. The Military Intel486 processor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment (i.e., if an operand has an offset greater than FFFFH, for example, a word with a low byte at FFFFH and the high byte at 0000H).

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64 Kbytes, another segment can be overlaid on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

5.3 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

5.4 Interrupts

Many of the exceptions shown in Table 4-16 and discussed in section 4.8.3, ‘‘Maskable Interrupt,’’ are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, 17, which do not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 5-2 identifies these exceptions.

5.5 Shutdown and Halt

The HALT instruction stops program execution and prevents the Military Intel486 processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the Military Intel486 processor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

As in the case of protected mode, the shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions, as follows:

- An interrupt or an exception occurs (exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e., there is not an interrupt handler for the interrupt).
- A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even (i.e., pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH).

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e., SP is greater than 0005H). If these conditions are not met, the Military Intel486 processor is unable to execute the NMI and executes another shutdown cycle. In this case, the Military Intel486 processor remains in the shutdown and can only exit via the RESET input.

Table 5-2. Exceptions with Different Meanings in Real Mode (see Table 4-17)

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

6.0 PROTECTED MODE ARCHITECTURE

The complete capabilities of the Military Intel486 processor are unlocked when the Military Intel486 processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four Gbytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the Military Intel486 processor to run all of the existing 8086, 80286 and Intel386 processor software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the Military Intel486 processor remains the same, the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode and Real Mode from a programmer's view is the increased address space and a different addressing mechanism.

6.1 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table. (See Figure 6-1.) The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the Military Intel486 processor. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 6-2 shows the complete Military Intel486 processor addressing mechanism with paging enabled.

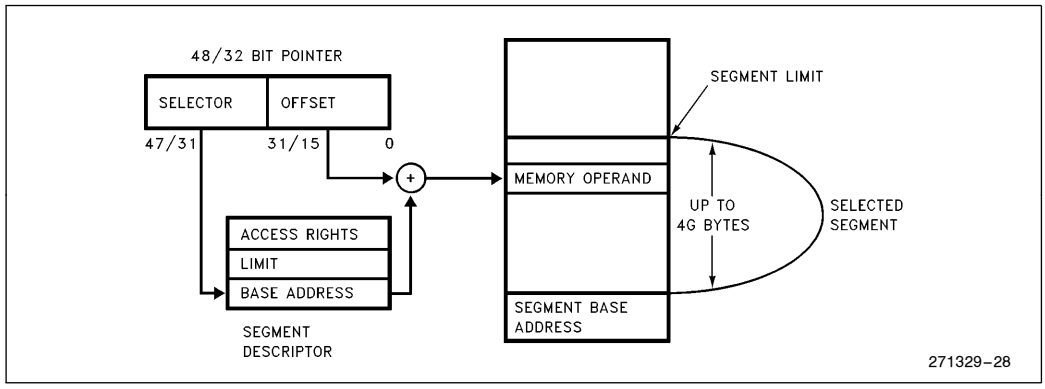


Figure 6-1. Protected Mode Addressing

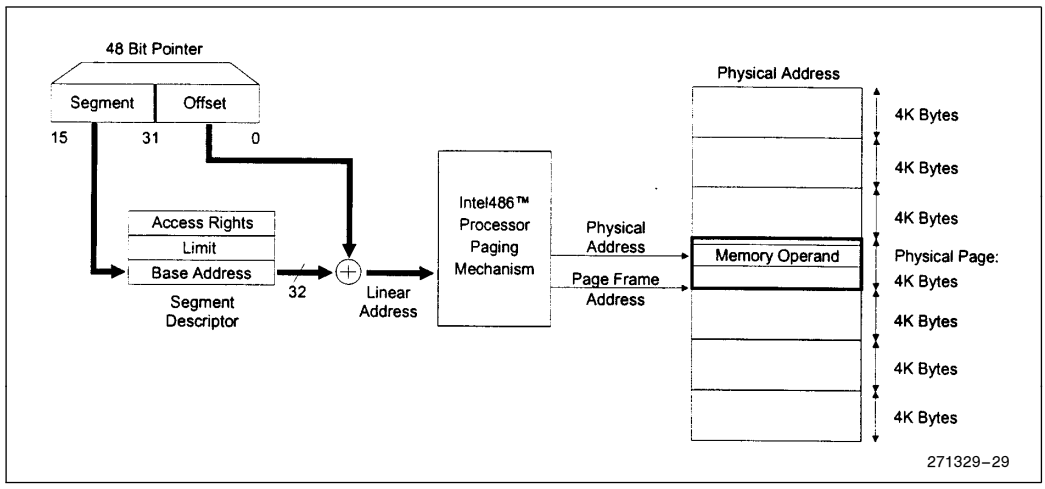


Figure 6-2. Paging and Segmentation

6.2 Segmentation

6.2.1 SEGMENTATION INTRODUCTION

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8-byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

6.2.2 TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requester Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Because smaller privilege level **values** indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

6.2.3 DESCRIPTOR TABLES

6.2.3.1 Descriptor Tables Introduction

The descriptor tables define all of the segments which are used in a Military Intel486 processor system. (See Figure 6-3.) There are three types of tables on the Military Intel486 processor which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it, the GDTR, LDTR, and the IDTR (see Figure 6-3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

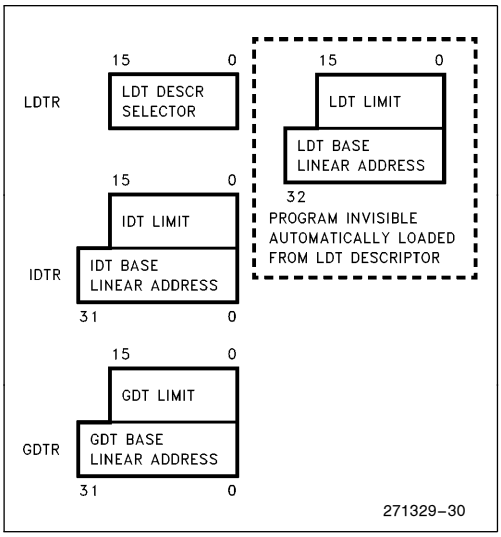


Figure 6-3. Descriptor Table Registers

6.2.3.2 Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e., interrupt and trap descriptors). Every Military Intel486 processor system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

6.2.3.3 Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6-byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

6.2.3.4 Interrupt Descriptor Table

The third table needed for Military Intel486 processor systems is the Interrupt Descriptor Table. (See Figure 6-4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See section 4.8, ''Interrupts.'')

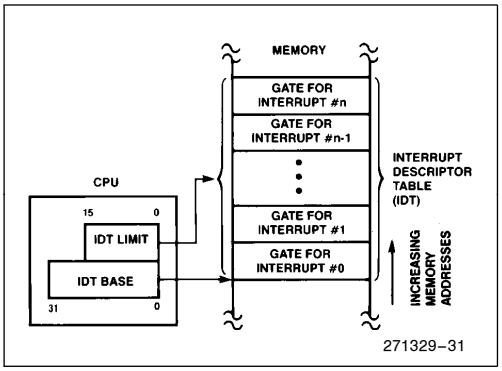


Figure 6-4. Interrupt Descriptor Table Register Use

6.2.4 DESCRIPTORS

6.2.4.1 Descriptor Attribute Bits

The object to which the segment selector points to is called a descriptor. Descriptors are eight-byte quantities that contain attributes about a given region of linear address space (i.e., a segment). These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. All segments on the Military Intel486 processor have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory. If P = 0, any attempt to access this segment will cause a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field that specifies the protection level 0–3 associated with a segment.

The Military Intel486 processor has two main categories of segments: system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the **S** bit is 1, the segment is either a code or data segment. If it is 0, the segment is a system segment.

6.2.4.2 Military Intel486 Processor Code, Data Descriptors (S = 1)

Figure 6-5 shows the general format of a code and data descriptor and Table 6-1 illustrates how the bits in the Access Rights Byte are interpreted. The Access Rights Bytes is bits 24–31 associated with the segment limit.

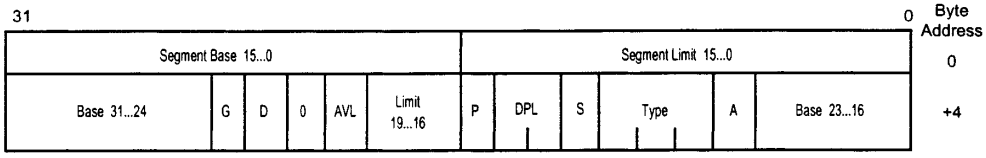
Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. Military Intel486 processor segments can be one megabyte long with byte granularity (G = 0) or four gigabytes with page granularity (G = 1), (i.e., 2^{20} pages each page is 4 Kbytes in length). The granularity is totally unrelated to paging. A Military Intel486 processor system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment (E = 1, S = 1) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if R = 0, and execute/read if R = 1. Code segments may never be written into.

NOTE:
Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If D = 1 then 32-bit operands and 32-bit addressing modes are assumed. If D = 0 then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the Military Intel486 processor assuming the D bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, C = 1, can be executed and shared by programs at different privilege levels. (See section 6.3, ''Protection.'')



BASE	Base Address of the segment
LIMIT	The length of the segment
P	Present Bit 1 = Present, 0 = Not Present
DPL	Descriptor Privilege Level 0–3
S	Segment Descriptor 0 = System Descriptor, 1 = Code or Data Segment Descriptor
TYPE	Type of Segment
A	Accessed Bit
G	Granularity Bit 1 = Segment length is page granular, 0 = Segment length is byte granular
D	Default Operation Size (recognized in code segment descriptors only) 1 = 32-bit segment, 0 = 16-bit segment
0	Bit must be zero (0) for compatibility with future processors
AVL	Available field for user or OS

NOTE:
In a maximum-size segment (i.e., a segment with G = 1 and segment limit 19. . .0 = FFFFFH), the lowest 12 bits of the segment base should be zero (i.e., segment base 11. . .000 = 000H).

Figure 6-5. Segment Descriptors

Table 6-1. Access Rights Byte Definition for Code and Data Descriptions

Bit Position	Name	Function	
7	Present (P)	P = 1	Segment is mapped into physical memory.
		P = 0	No mapping to physical memory exits, base and limit are not used.
6–5	Descriptor Privilege Level (DPL)		Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	S = 1	Code or Data (includes stacks) segment descriptor.
		S = 0	System Segment Descriptor or Gate Descriptor.
		If Data Segment (S = 1, E = 0)	
3	Executable (E)	E = 0	Descriptor type is data segment:
2	Expansion Direction (ED)	ED = 0	Expand up segment, offsets must be ≤ limit.
		ED = 1	Expand down segment, offsets must be > limit.
1	Writeable (W)	W = 0	Data segment may not be written into.
		W = 1	Data segment may be written into.
		If Code Segment (S = 1, E = 1)	
3	Executable (E)	E = 1	Descriptor type is code segment:
2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.
1	Readable (R)	R = 0	Code segment may not be read.
		R = 1	Code segment may be read.
0	Accessed (A)	A = 0	Segment has not been accessed.
		A = 1	Segment selector has been loaded into segment register or used by selector test instructions.

Segments identified as data segments (E = 0, S = 1) are used for two types of Military Intel486 processor segments: stack and data segments. The expansion direction **(ED)** bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if W = 0. The stack segment must have W = 1.

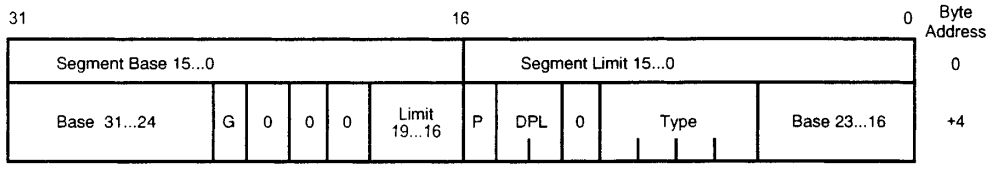
The **B** bit controls the size of the stack pointer register. If B = 1, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If B = 0, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

6.2.4.3 System Descriptor Formats

System segments describe information about operating system tables, tasks, and gates. Figure 6-6 shows the general format of system segment descriptors, and the various types of system segments. Military Intel486 processor system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

6.2.4.4 LDT Descriptors (S = 0, TYPE = 2)

LDT descriptors (S = 0, TYPE = 2) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Because the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).



Type	Defines
0	Invalid
1	Available 80286 TSS
2	LDT
3	Busy 80286 TSS
4	80286 call gate
5	Task Gate (for 80286, Military Intel486™ processor task)
6	80286 interrupt gate
7	80286 trap gate
8	Invalid
9	Available Military Intel486 processor TSS
A	Undefined (Intel Reserved)
B	Busy Military Intel486 processor TSS
C	Military Intel486 processor call gate
D	Undefined (Intel Reserved)
E	Military Intel486 processor
F	Military Intel486 processor

Figure 6-6. System Segment Descriptors

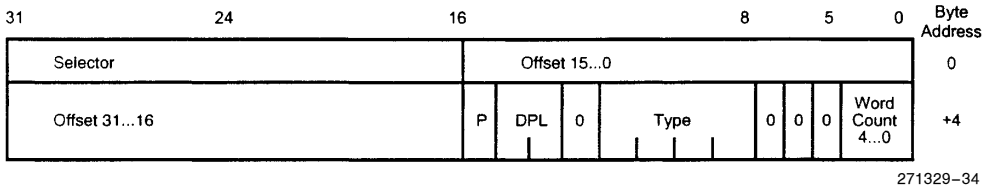
6.2.4.5 TSS Descriptors (S = 0, TYPE = 1, 3, 9, B)

A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e., on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains an 80286 processor TSS or a Military Intel486 processor TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

6.2.4.6 Gate Descriptors (S = 0, TYPE = 4–7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of gate descriptors are **call** gates, **task** gates, **interrupt** gates, and **trap** gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see section 6.3, ''Protection''), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.

Figure 6-7 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.



Gate Descriptor Fields

Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or Military Intel486 processor task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	Military Intel486™ processor call gate
	E	Military Intel486 processor interrupt gate
	F	Military Intel486 processor trap gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid

DPL—least privileged level at which a task may access the gate. WORD COUNT 0–31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for Military Intel486 processor gates, and 16-bit quantities for 80286 gates.

DESTINATION SELECTOR	16-bit selector	Selector to the target code segment or Selector to the target task state segment for task gate
DESTINATION OFFSET	offset	Entry point within the target code segment 16-bit 80286 32-bit Military Intel486 processor

Figure 6-7. Gate Descriptor Formats

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit), while the trap gate does not.

Task gates are used to switch tasks. Task gates may only refer to a task state segment. (See section 6.3.6, ''Task Switching.'') Therefore, only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

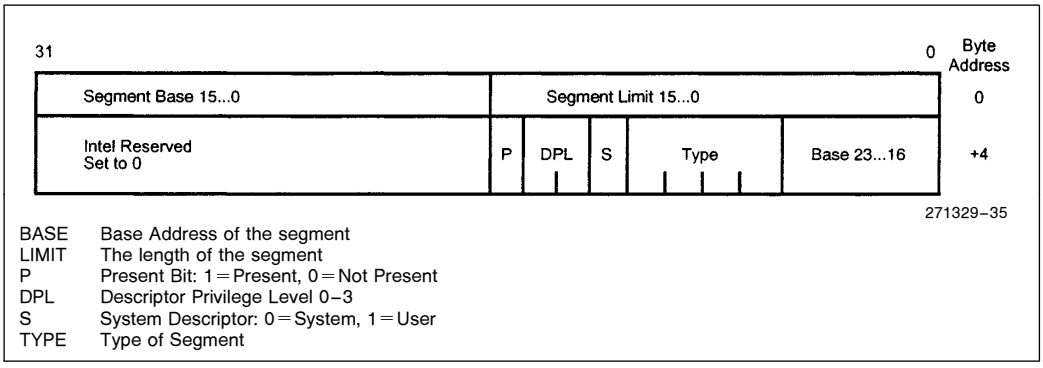
The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task. (See section 6.3, ''Protection.'') The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 6-7.

6.2.4.7 Differences Between Military Intel486 Processor and 80286 Descriptors

In order to provide operating system compatibility between 80286 and Military Intel486 processors, the Military Intel486 processor supports all of the 80286 segment descriptors. Figure 6-8 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and Military Intel486 processor descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the Military Intel486 processor. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the Military Intel486 processor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.

By supporting 80286 system segments the Military Intel486 processor is able to execute 80286 application programs on a Military Intel486 processor operating system. This is possible because the Military Intel486 processor automatically understands which descriptors are 80286-style descriptors and which descriptors are Military Intel486 processor-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is an 80286-style descriptor.

The only other differences between 80286-style descriptors and Military Intel486 processor descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for Military Intel486 processor call gates. The B bit controls the size of PUSHes when using a call gate; if B = 0 PUSHes are 16 bits, if B = 1 PUSHes are 32 bits.



BASE Base Address of the segment
LIMIT The length of the segment
P Present Bit: 1 = Present, 0 = Not Present
DPL Descriptor Privilege Level 0–3
S System Descriptor: 0 = System, 1 = User
TYPE Type of Segment

Figure 6-8. 80286 Code and Data Segment Descriptors

6.2.4.8 Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requester (the selector's) Privilege Level (RPL) as shown in Figure 6-9. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

6.2.4.9 Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Because descriptor caches only change when a segment register is changed, programs that modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

6.2.4.10 Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the Military Intel486 processor is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 6-10. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal ''privilege level'' is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.

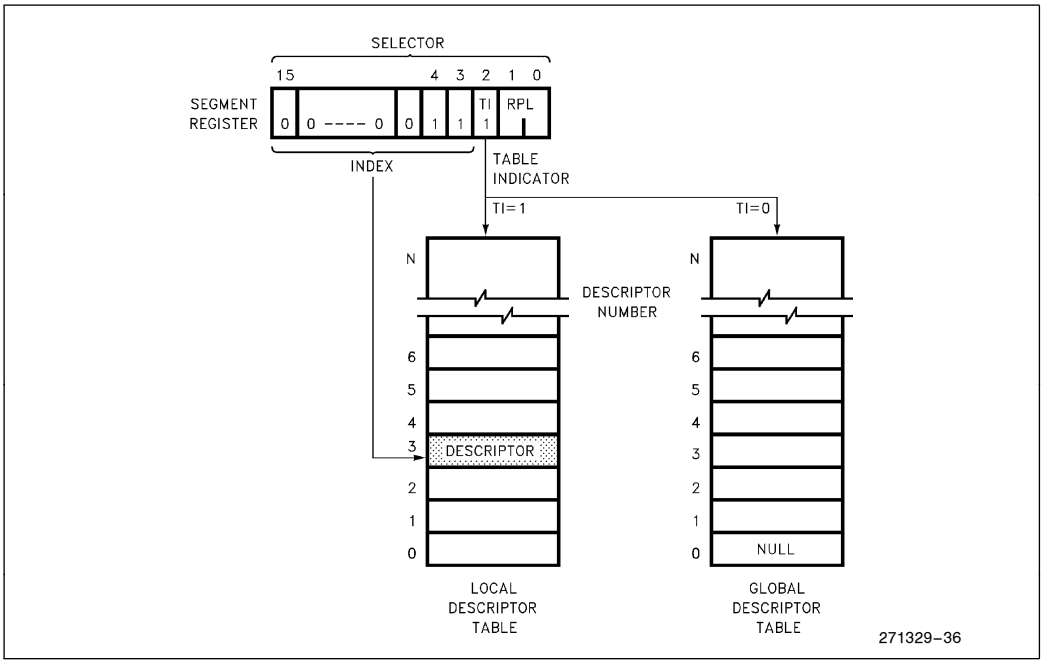
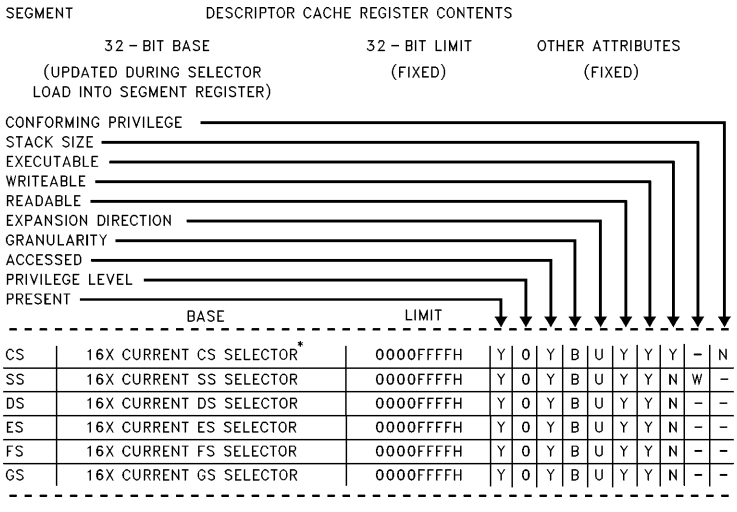


Figure 6-9. Example Descriptor Selection



*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (i.e., intersegment CALL, or intersegment JMP, or INT). (See Figure 6-12 for an example.)

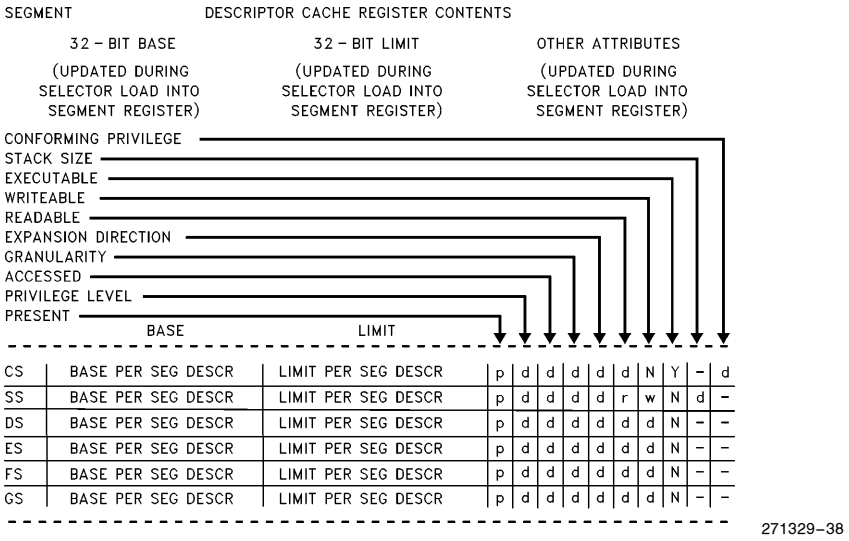
Key:

Y = yes	D = expand down
N = no	B = byte granularity
0 = privilege level 0	P = page granularity
1 = privilege level 1	W = push/pop 16-bit words
2 = privilege level 2	F = push/pop 32-bit dwords
3 = privilege level 3	— = does not apply to that segment cache register
U = expand up	

Figure 6-10. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes Are Fixed)

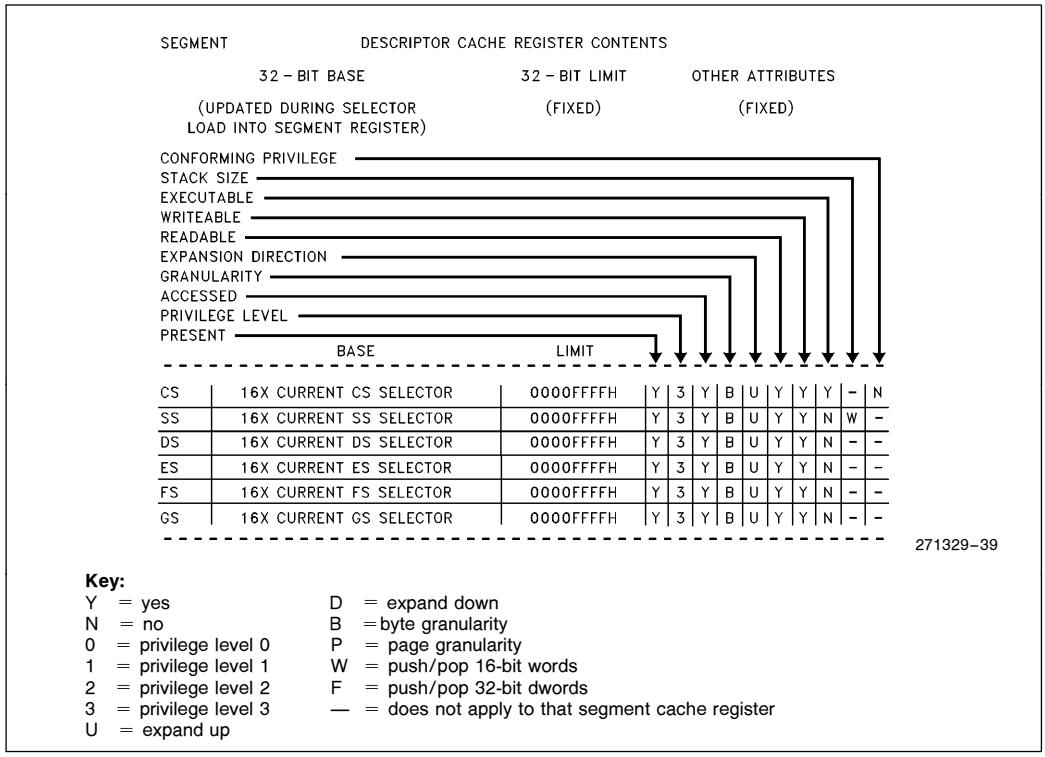
When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 6-11. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 6-12. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.



Key:
Y = fixed yes
N = fixed no
d = per segment descriptor
p = per segment descriptor; descriptor must indicate "present" to avoid exception 11 (exception 12 in case of SS)
r = per segment descriptor, but descriptor must indicate "readable" to avoid exception 13 (special case for SS)
w = per segment descriptor, but descriptor must indicate "writeable" to avoid exception 13 (special case for SS)
— = does not apply to that segment cache register

Figure 6-11. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)



Key:

Y = yes	D = expand down
N = no	B = byte granularity
0 = privilege level 0	P = page granularity
1 = privilege level 1	W = push/pop 16-bit words
2 = privilege level 2	F = push/pop 32-bit dwords
3 = privilege level 3	— = does not apply to that segment cache register
U = expand up	

Figure 6-12. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

6.3 Protection

6.3.1 PROTECTION CONCEPTS

The Military Intel486 processor has four levels of protection which are optimized to support the needs of a multitasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional processor-based systems where this protection is achieved only through the use of complex external hardware and software the Military Intel486 processor provides the protection as part of its integrated Memory Management Unit. The Military Intel486 processor offers an additional type of protection on a page basis, when paging is enabled (See section 6.4.3, ''Page Level Protection.'')

The four-level hierarchical privilege system is illustrated in Figure 6-13. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is fully supported by the Military Intel486 processor paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

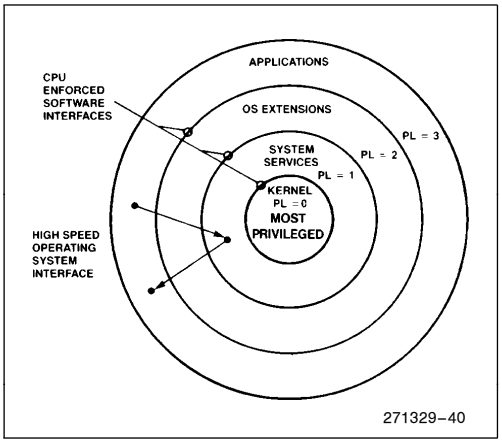


Figure 6-13. Four-Level Hierarchical Protection

6.3.2 RULES OF PRIVILEGE

The Military Intel486 processor controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

6.3.3 PRIVILEGE LEVELS

6.3.3.1 Task Privilege

At any point in time, a task on the Military Intel486 processor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See section 6.3.4, ''Privilege Level Transfers.'') Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

6.3.3.2 Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e., numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Because the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

6.3.3.3 I/O Privilege and I/O Permission Bitmap

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when CPL $\geq$ IOPL. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When CPL $>$ IOPL, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When CPL $>$ IOPL, and the current task is associated with a Military Intel486 processor TSS, the I/O Permission Bitmap (part of a Military Intel486 processor TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bitmap, refer to Figure 6-14 and Figure 6-15. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to section 6.5.4, ''Protection and I/O Permission Bitmap.''

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called ''IOPL-sensitive'' instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the Military Intel486 processor.)

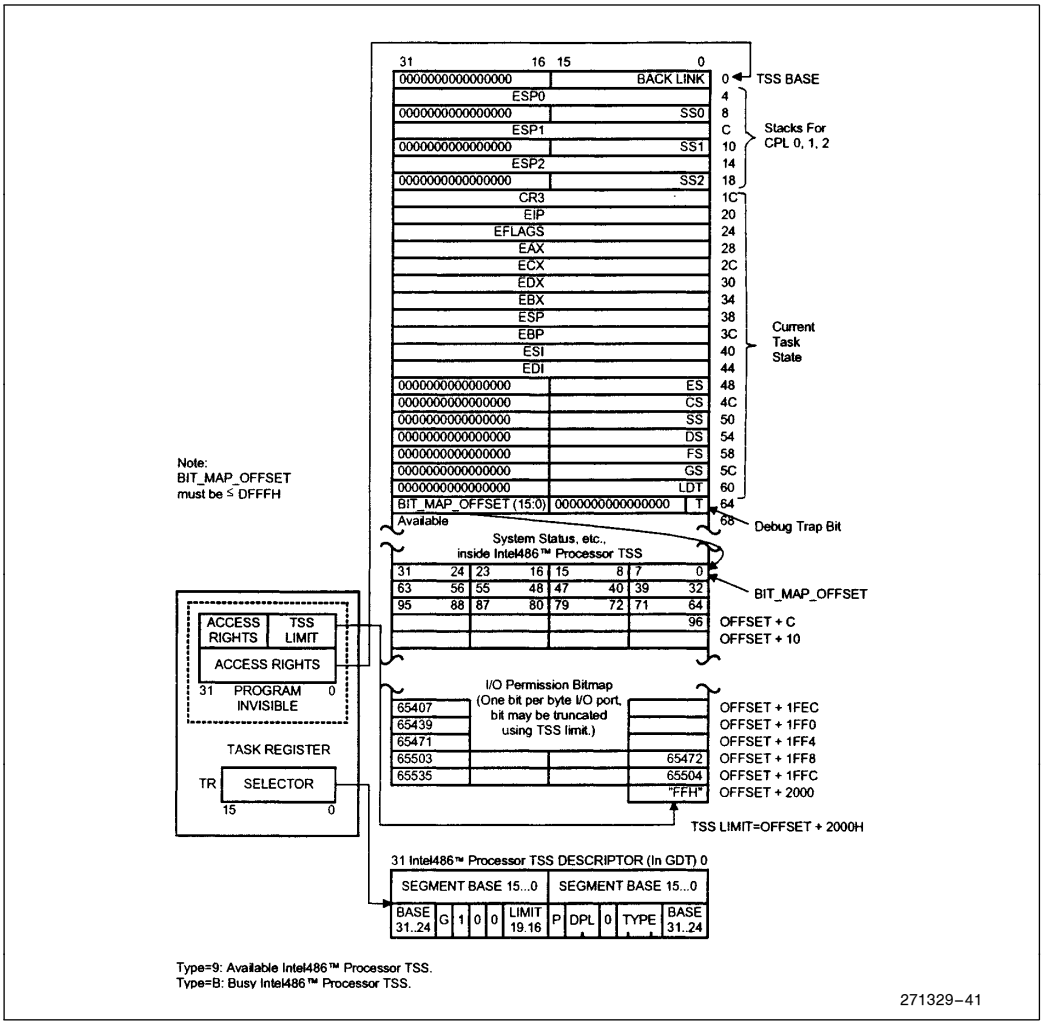
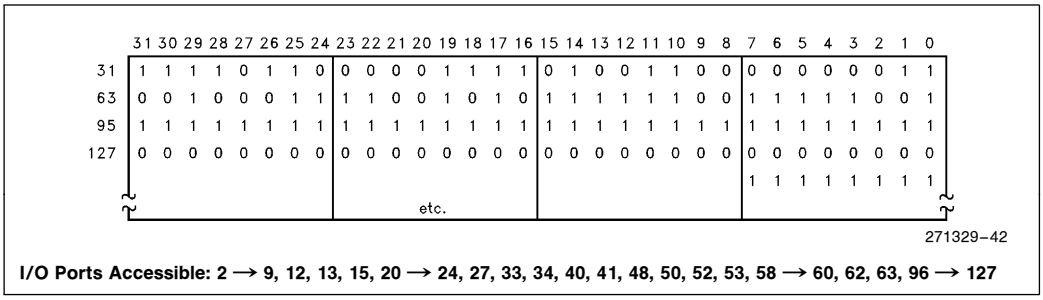


Figure 6-14. Military Intel486™ Processor TSS and TSS Registers



I/O Ports Accessible: 2 ⟶ 9, 12, 13, 15, 20 ⟶ 24, 27, 33, 34, 40, 41, 48, 50, 52, 53, 58 ⟶ 60, 62, 63, 96 ⟶ 127

Figure 6-15. Sample I/O Permission Bit Map

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When CPL ≥ IOPL, then the IF bit can be changed by loading a new value into the EFLAGS register. When CPL > IOPL, the IF bit cannot be changed by a new value POPed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

6.3.3.4 Privilege Validation

The Military Intel486 processor provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 6-2 summarizes the selector validation procedures available for the Military Intel486 processor.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a ''bad'' pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

6.3.3.5 Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Table 6-2. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the Military Intel486 processor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in section 6.3.2, ''Rules of Privilege.'' The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

6.3.4 PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 6-3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g., JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

Table 6-3. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level	CALL	Call Gate	GDT/LDT
Interrupt within task may change CPL	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET(1)	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET(2) Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

NOTES:
1. NT (Nested Task bit of flag register) = 0
2. NT (Nested Task bit of flag register) = 1

The privilege rules require that:

- Privilege level transitions can only occur via gates.
- JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.
- CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.
- Interrupts handled within the task obey the same privilege rules as CALLs.
- Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.
- Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.
- The code segment selected in the gate must be the same or more privileged than the task's CPL.
- Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.
- Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment. (See section 6.3.6, ''Task Switching.'') During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETurning to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

6.3.5 CALL GATES

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Because the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level Military Intel486 processor call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e., the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

6.3.6 TASK SWITCHING

A very important attribute of any multitasking/multiuser operating system is its ability to rapidly switch between tasks or processes. The Military Intel486 processor directly supports this operation by providing a task switch instruction in hardware. The Military Intel486 processor task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks,

and commences execution in the new task, in about 10 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 6-14) containing the entire Military Intel486 processor execution state while a task gate descriptor contains a TSS selector. The Military Intel486 processor supports both 80286 and Military Intel486 processor style TSSs. Figure 6-16 shows an 80286 TSS. The limit of a Military Intel486 processor TSS must be greater than 0064H (002BH for an 80286 TSS), and can be as large as 4 Gbytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

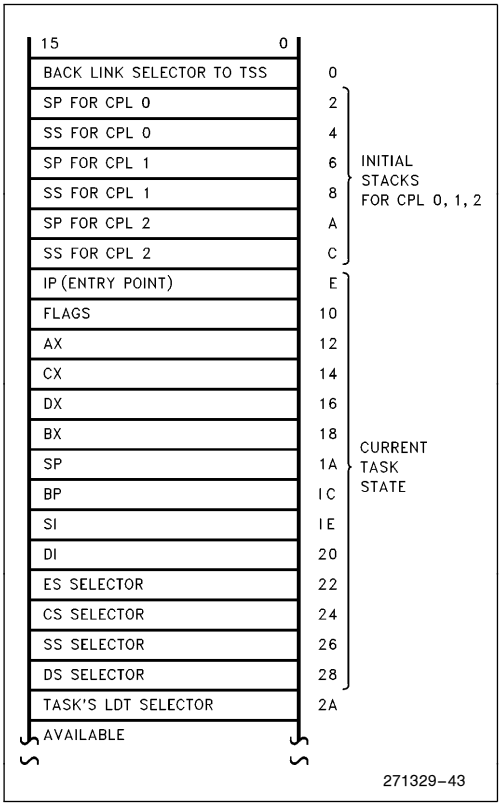


Figure 6-16. 80286 TSS

Each task must have a TSS associated with it. The current TSS is identified by a special register in the Military Intel486 processor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The Military Intel486 processor task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch. (See section 6.5, "Virtual 8086 Environment.")

The **T** bit in the Military Intel486 processor TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1, upon entry to a new task, a debug exception 1 will be generated.

6.3.6.1 Floating Point Task Switching

The FPU's state is not automatically saved when a task switch occurs, because the incoming task may not use the FPU. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the FPU's state in a multitasking environment. Whenever the Intel OverDrive processors switch tasks, they set the TS bit. The Intel OverDrive processors detect the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the FPU. A processor extension not present exception (7) will occur when attempting to execute a Floating Point or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e., TS = 1 and MP = 1).

6.3.7 INITIALIZATION AND TRANSITION TO PROTECTED MODE

Because the Military Intel486 processor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256-bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 6-17 shows the tables and Figure 6-18 the descriptors needed for a simple Protected Mode Military Intel486 processor system. It has a single code and single data/stack segment each four-Gbytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the Military Intel486 processor in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

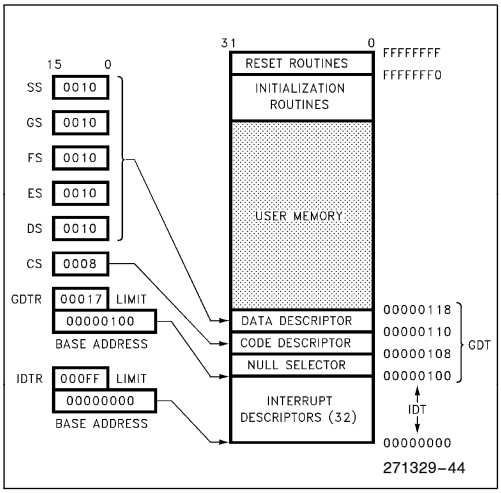


Figure 6-17. Simple Protected System

An alternate approach to entering Protected Mode which is especially appropriate for multitasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. Because a task switch saves the state of the current task in a task state segment, the Task State Segment Register should be initialized to point to a valid TSS descriptor.

6.4 Paging

6.4.1 PAGING CONCEPTS

Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical structure of a program. While segment selectors can be considered the logical ''name'' of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

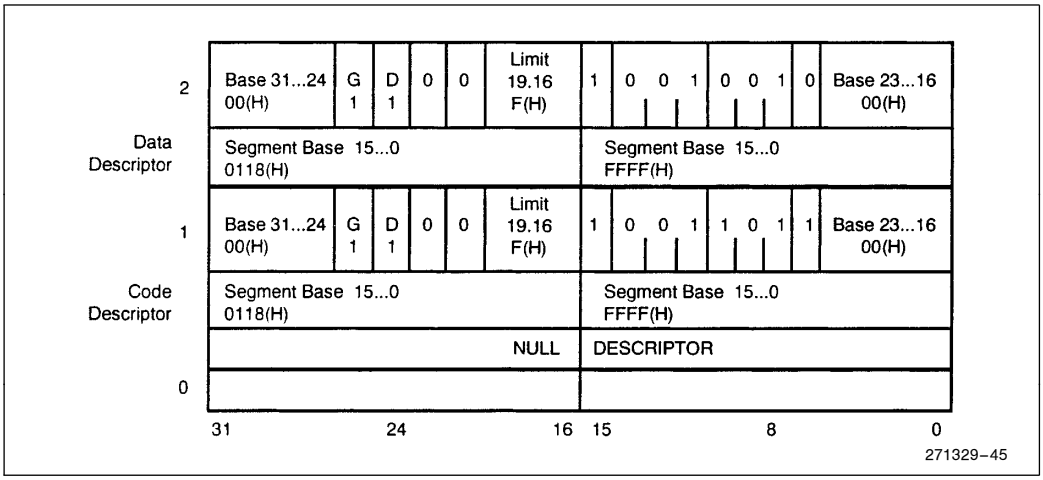


Figure 6-18. GDT Descriptors for Simple System

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

6.4.2 PAGING ORGANIZATION

6.4.2.1 Page Mechanism

The Military Intel486 processor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the Military Intel486 processor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the Military Intel486 processor paging mechanism are the same size, namely, 4 Kbytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, because there is no problem with memory fragmentation. Figure 6-19 shows how the paging mechanism works.

6.4.2.2 Page Descriptor Base Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3 reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS that **changes** the value of CR0. (See section 6.4.5, "Translation Lookaside Buffer.")

6.4.2.3 Page Directory

The Page Directory is 4-Kbytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 6-20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

6.4.2.4 Page Tables

Each Page Table is 4 Kbytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page. (See Figure 6-21.) Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

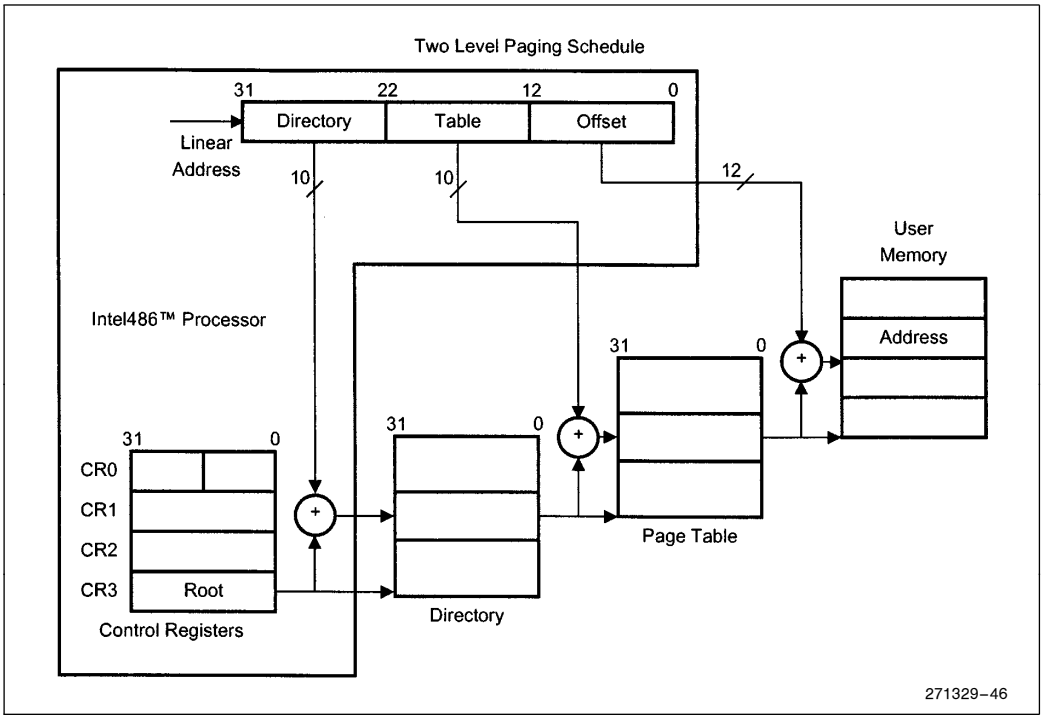


Figure 6-19. Paging Mechanism

31 — 12	11 — 10	9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31 . . . 12	OS RESERVED		0	0	D	A	PCD	PWT	U—S	R—W	P

Figure 6-20. Page Directory Entry (Points to Page Table)

31 — 12	11 — 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31 . . . 12	OS RESERVED		0	0	D	A	PCD	PWT	U—S	R—W	P

Figure 6-21. Page Table Entry (Points to Page)

6.4.2.5 Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation, if P = 0 the entry cannot be used for translation, and all of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the Military Intel486 processor for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the Military Intel486 processor, a Read-Modify-Write cycle is generated which locks the bus and prevents conflicts with other processors or peripherals. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multimaster systems.

The 3 bits marked **OS Reserved** in Figure 6-20 and Figure 6-21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm such as Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

6.4.3 PAGE LEVEL PROTECTION (R/W, U/S BITS)

The Military Intel486 processor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2).

The R/W and U/S bits are used in conjunction with the WP bit in the flags register (EFLAGS). The Intel386 processor does not contain the WP bit. The WP bit has been added to the Military Intel486 processor to protect read-only pages from supervisor write accesses. The Intel386 processor allows a read-only page to be written from protection levels 0, 1 or 2. WP = 0 is the Intel386 processor compatible mode. When WP = 0, the supervisor can write to a read-only page as defined by the U/S and R/W bits. When WP = 1, supervisor access to a read-only page (R/W = 0) will cause a page fault (exception 14).

Table 6-4 shows the affect of the WP, U/S and R/W bits on accessing memory. When WP = 0, the supervisor can write to pages regardless of the state of the R/W bit. When WP = 1 and R/W = 0, the supervisor cannot write to a read-only page. A user attempt to access a supervisor only page (U/S = 0) or to write to a read-only page will cause a page fault (exception 14).

Table 6-4. Page Level Protection Attributes

U/S	R/W	WP	User Access	Supervisor Access
0	0	0	None	Read/Write/Execute
0	1	0	None	Read/Write/Execute
1	0	0	Read/Execute	Read/Write/Execute
1	1	0	Read/Write/Execute	Read/Write/Execute
0	0	1	None	Read/Execute
0	1	1	None	Read/Write/Execute
1	0	1	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

The R/W and U/S bits provide protection from user access on a page by page basis because the bits are contained in the Page Table Entry and the Page Directory Table. The U/S and R/W bits in the first-level Page Directory Table apply to all entries in the page table pointed to by that directory entry. The U/S and R/W bits in the second-level Page Table Entry apply only to the page described by that entry. The most restrictive of the U/S and R/W bits from the Page Directory Table and the Page Table Entry are used to address a page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 (user read/execute) and the U/S and R/W bits for the Page Table Entry were 01 (no user access at all), the access rights for the page would be 01, the numerically smaller of the two.

Note that a given segment can be easily made read-only for level 0, 1 or 2 via use of segmented protection mechanisms. (Section 6.3, ‘‘Protection’’.)

6.4.4 PAGE CACHEABILITY (PWT AND PCD BITS)

See section 7.6, ‘‘Page Cacheability,’’ for a detailed description of page cacheability and the PWT and PCD bits.

6.4.5 TRANSLATION LOOKASIDE BUFFER

The Military Intel486 processor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the Military Intel486 processor was required to access two levels of tables for every memory reference. To solve this problem, the Military Intel486 processor keeps a cache of the most recently accessed pages. This cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the Military Intel486 processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128 Kbytes of memory addresses. For many common multitasking systems, the TLB will have a hit rate of about 98%. This means that the Military Intel486 processor will only have to access the two-level page structure on 2% of all memory references. Figure 6-22 illustrates how the TLB complements the Military Intel486 processor’s paging mechanism.

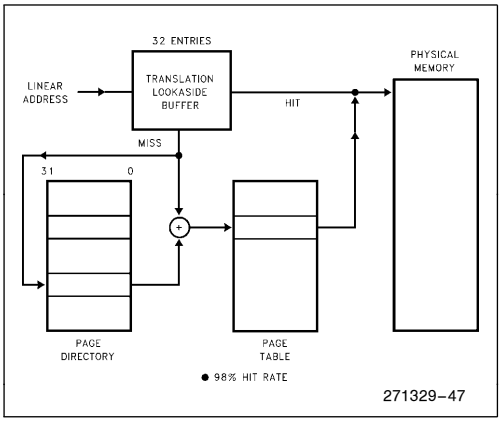


Figure 6-22. Translation Lookaside Buffer

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the Military Intel486 processor hardware. The sequence of data cycles to perform a TLB refresh are the following:

1. Read the correct Page Directory Entry, as pointed to by the page base register and the upper 10 bits of the linear address. The page base register is in control register 3.
   a. Optionally perform a locked read/write to set the accessed bit in the directory entry. The directory entry will actually get read twice if the Military Intel486 processor needs to set any of the bits in the entry. If the page directory entry changes between the first and second reads, the data returned for the second read will be used.
2. Read the correct entry in the Page Table and place the entry in the TLB.
   a. Optionally perform a locked read/write to set the accessed and/or dirty bit in the page table entry. Again, note that the page table entry will actually get read twice if the Military Intel486 processor needs to set any of the bits in the entry. Like the directory entry, if the data changes between the first and second read the data returned for the second read will be used.

Note that the directory entry must always be read into the Military Intel486 processor, because directory entries are never placed in the paging TLB. Page faults can be signaled from either the page directory read or the page table read. Page directory and page table entries may be placed in the Military Intel486 processor on-chip cache just like normal data.

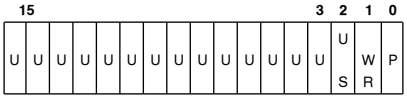
6.4.6 PAGING OPERATION

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e., a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the Military Intel486 processor will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry indicating that the page table is in memory, then the Military Intel486 processor will read the appropriate Page Table Entry and set the Access bit. If P = 1 on the Page Table Entry indicating that the page is in memory, the Military Intel486 processor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory Entry or the Page Table Entry, then the Military Intel486 processor will generate a page fault, an Exception 14.

The Military Intel486 processor will also generate an exception 14 page fault if the memory reference violated the page protection attributes (i.e., U/S or R/W) (e.g., trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the Military Intel486 processor is attempting to enter the service routine for the first, then the Military Intel486 processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Because Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault. The upper portion of Figure 6-23 shows the format of the page-fault error code and the interpretation of the bits.



U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

NOTES:
*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

U: UNDEFINED

U/S: The U/S bit indicates whether the access causing the fault occurred when the Military Intel486 processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0).

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1).

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1).

Figure 6-23. Page Fault System Information

NOTE:
Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 6-23 indicates what type of access caused the page fault.

6.4.7 OPERATING SYSTEM RESPONSIBILITIES

The Military Intel486 processor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for

setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e., flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

6.5 Virtual 8086 Environment

6.5.1 EXECUTING 8086 PROGRAMS

The Military Intel486 processor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the Military Intel486 processor protection mechanism. In particular, the Military Intel486 processor allows the simultaneous execution of 8086 operating systems and its applications, and a Military Intel486 processor operating system and both 80286 and Military Intel486 processor applications. Thus, in a multi-user Military Intel486 processor computer, one person could be running an MS-DOS* spreadsheet, another person using MS-DOS*, and a third person could be running multiple UNIX utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 6-24 illustrates this concept.

6.5.2 VIRTUAL 8086 MODE ADDRESSING MECHANISM

One of the major differences between Military Intel486 processor Real and Protected modes is how the segment selectors are interpreted. When the Military Intel486 processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The Military Intel486 processor allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4-Gbyte linear address space of the Military Intel486 processor. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 Kbyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

6.5.3 PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one Mbyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4-Gbyte physical address space of the Military Intel486 processor. In addition, because CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 6-24 shows how the Military Intel486 processor paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

94

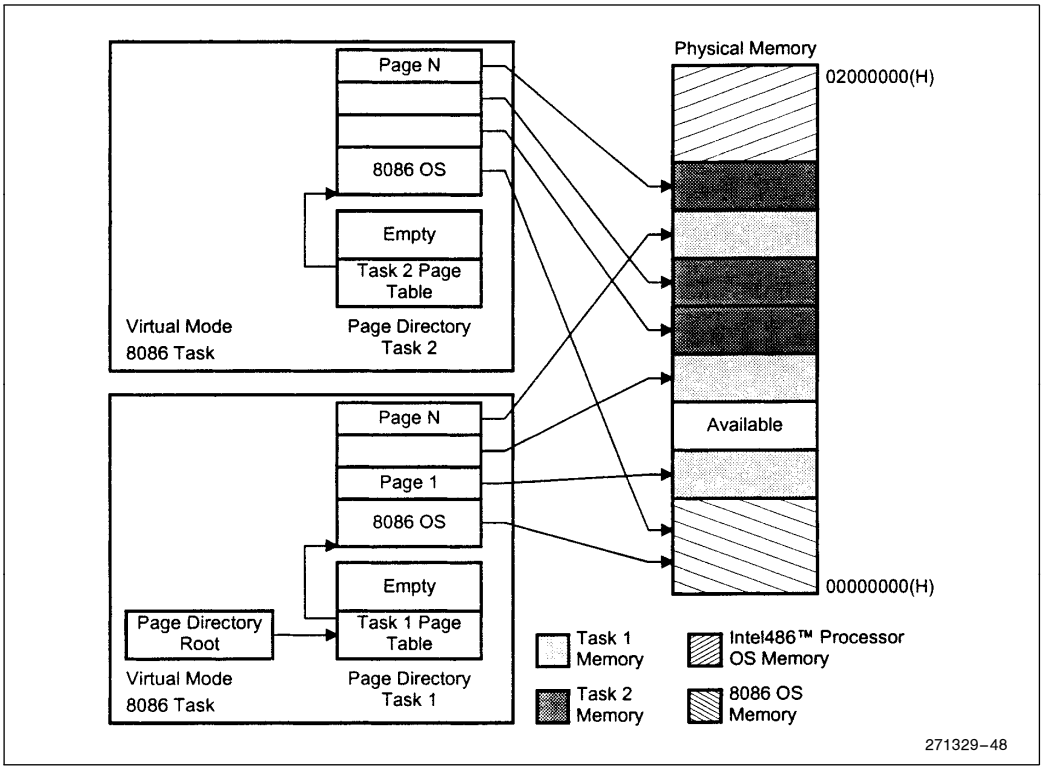


Figure 6-24. Virtual 8086 Environment Memory Management

6.5.4 PROTECTION AND I/O PERMISSION BITMAP

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL $>$ 0) causes an exception 13 fault:

```
LIDT; MOV DRn,reg; MOV reg,DRn;
LGDT; MOV TRn,reg; MOV reg,TRn;
LMSW; MOV CRn,reg; MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;      STR;
LLDT;     SLDT;
LAR;      VERR;
LSL;      VERW;
ARPL.
```

The instructions that are IOPL-sensitive in Protected Mode are:

```
IN;       STI;
OUT;      CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;    STI;
PUSHF;    CLI;
POPF;     IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **Military Intel486 processor Task State Segment**. The I/O Permission Bitmap, automatically used by the Military Intel486 processor in Virtual 8086 Mode, is illustrated by Figure 6-14 and Figure 6-15.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit string, which begins in memory at offset Bit__Map__Offset in the current TSS. Bit__Map__Offset must be ≤ DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets ≤ FFFFH from the TSS base. The 16-bit pointer Bit__Map__Offset (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 6-14.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 6-14. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Because every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the Bit__Map__Offset (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

Example of Bitmap for I/O Ports 0–255: Setting the TSS limit to {bit__Map__Offset + 31 + 1**} [** see note below] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [** see note below]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

> ****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the Military Intel486 processor TSS segment (see Figure 6-14).

6.5.5 INTERRUPT HANDLING

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host Military Intel486 processor operating system. The Military Intel486 processor operating system determines if the interrupt

comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The Military Intel486 processor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The Military Intel486 processor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the Military Intel486 processor operating system. The Military Intel486 processor operating system could emulate the 8086 operating system's call. Figure 6-25 shows how the Military Intel486 processor operating system could intercept an 8086 operating system's call to ''Open a File.''

A Military Intel486 processor operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

6.5.6 ENTERING AND LEAVING VIRTUAL 8086 MODE

Virtual 8086 mode is entered by executing an IRET instruction (at CPL = 0), or Task Switch (at any CPL) to a Military Intel486 processor task whose Military Intel486 processor TSS has a FLAGS image containing a 1 in the VM bit position while the Military Intel486 processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with a Military Intel486 processor TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the Military Intel486 processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the Military Intel486 processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to Military Intel486 processor protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected Military Intel486 processor mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

6.5.6.1 Task Switches To and From Virtual 8086 Mode

Tasks which can execute in virtual 8086 mode must be described by a TSS with the new Military Intel486 processor format (TYPE 9 or 11 descriptor).

A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a Military Intel486 processor TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS.

The segment registers in the TSS will contain 8086 segment base values rather than selectors.

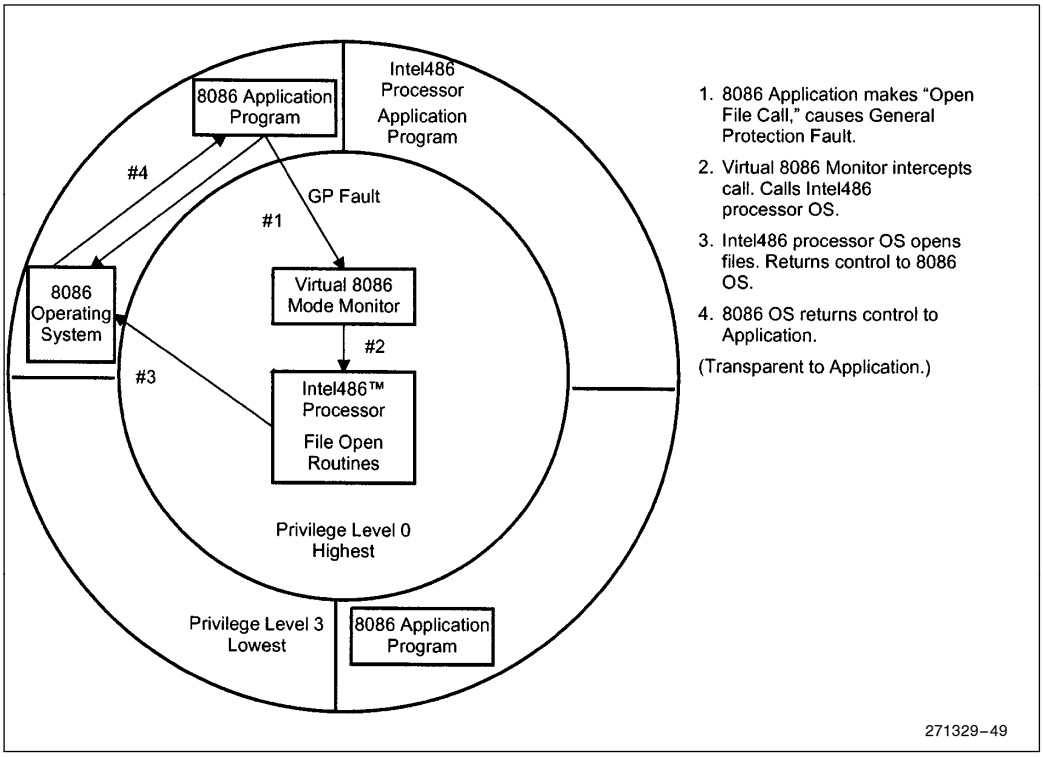


Figure 6-25. Virtual 8086 Environment Interrupt and Call Handling

A task switch into a task described by a Military Intel486 processor TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a Military Intel486 processor TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

6.5.6.2 Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a Military Intel486 processor Trap Gate (Type 14), or Military Intel486 processor Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. Military Intel486 processor gates must be used, because 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a Military Intel486 processor Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

1. Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.
2. Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, because the VM bit was turned off above.
3. Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).
4. Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.
5. Push the 32-bit FLAGS register saved in step 1.
6. Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.
7. Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected Military Intel486 processor mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e., push all registers in prolog, pop all in epilog) regardless of whether or not a 'native' mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended Military Intel486 processor IRET instruction (operand size = 32) can be used, and must be executed at level 0 to change the VM bit to 1.

1. If the NT bit in the FLAGs register is on, an intertask return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed. Otherwise, continue with the following sequence.
2. Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.
3. Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM = 0, this CS load is done as a protected mode segment load. If VM = 1, this will be done as an 8086 segment load.
4. ESP register by 4 to bypass the FLAGS image which was ''popped'' in step 1.
5. If VM = 1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP + 8], SS:[ESP + 12], SS:[ESP + 16], and SS:[ESP + 20], respectively, where the new value of ESP stored in step 4 is used. Because VM = 1, these are done as 8086 segment register loads. Else if VM = 0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.
6. If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM = 0, SS is loaded as a protected mode segment register load. If VM = 1, an 8086 segment register load is used.
7. Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the Military Intel486 processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

7.0 ON-CHIP CACHE

All members of the Military Intel486 processor family, except the IntelDX4 processor, contain an on-chip 8-Kbyte cache. (See section 7.1.1, ''IntelDX4 Processor Cache,'' for the IntelDX4 processor cache organization.) The cache is software transparent to maintain binary compatibility with previous generations of the Intel Architecture.

The on-chip cache has been designed for maximum flexibility and performance. The cache has several operating modes offering flexibility during program execution and debugging. Memory areas can be defined as non-cacheable by software and external hardware. Protocols for cache line invalidations and replacement are implemented in hardware, easing system design.

7.1 Cache Organization

The on-chip cache is a unified code and data cache. The cache is used for both instruction and data accesses and acts on physical addresses.

The cache organization is 4-way set associative and each line is 16-bytes wide. The eight Kbytes of cache memory are logically organized as 128 sets, each containing four lines.

The cache memory is physically split into four 2-Kbyte blocks, each containing 128 lines. (See Figure 7-1.) There are 128 21-bit tags associated with each 2-Kbyte block. There is a valid bit for each line in the cache. Each line in the cache is either valid or not valid. There are no provisions for partially valid lines.

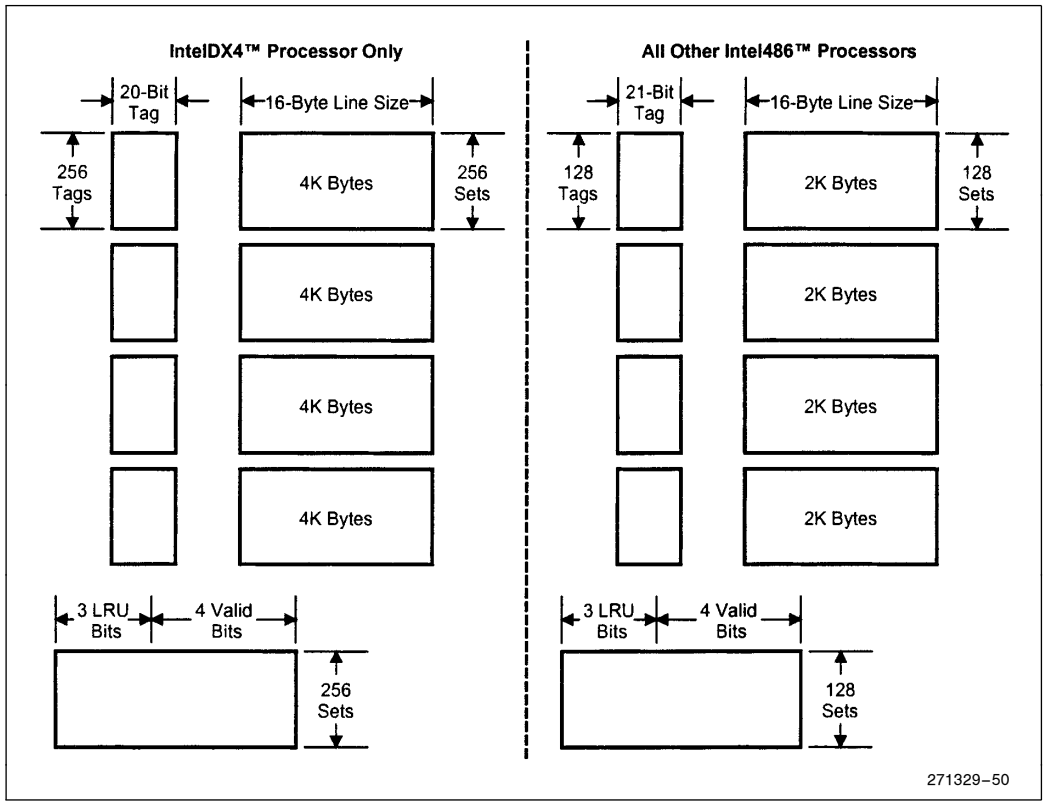


Figure 7-1. On-Chip Cache Physical Organization

For all Military Intel486 processors the on-chip cache is write-through only. All writes will drive an external write bus cycle in addition to writing the information to the internal cache if the write was a cache hit. A write to an address not contained in the internal cache will only be written to external memory. Cache allocations are not made on write misses.

7.1.1 INTELDX4 PROCESSOR CACHE

The IntelDX4 processor contains a 16-Kbyte write-through cache. The 16 Kbytes of cache memory are logically organized as 256 sets, each containing four lines.

The cache memory is physically split into four 4-Kbyte blocks, each containing 256 lines. (See Figure 7-1.) There are 256 20-bit tags associated with each 2-Kbyte block.

All other details listed in section 7.1 for the 8-Kbyte on-chip cache also apply to the IntelDX4 on-chip cache.

7.2 Cache Control

Control of the cache is provided by the CD and NW bits in CR0. CD enables and disables the cache. NW controls memory write-through and invalidates.

The CD and NW bits define four operating modes of the on-chip cache as given in Table 7-1. These modes provide flexibility in how the on-chip cache is used.

CD = 1, NW = 1

The cache is completely disabled by setting CD = 1 and NW = 1 and then flushing the cache. This mode may be useful for debugging programs where it is important to see all memory cycles at the pins. Writes that hit in the cache will not appear on the external bus.

It is possible to use the on-chip cache as fast static RAM by ''pre-loading'' certain memory areas into the cache and then setting CD = 1 and NW = 1. Pre-loading can be done by careful choice of memory references with the cache turned on or by use of the testability functions. (See section 11.2, ''On-Chip Cache Testing.'') When the cache is turned off, the memory mapped by the cache is ''frozen'' into the cache because fills and invalidates are disabled.

Table 7-1. Cache Operating Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled
1	0	Cache fills disabled, write-through and invalidates enabled
0	1	INVALID. If CR0 is loaded with this configuration of bits, a GP fault with error code of 0 is raised.
0	0	Cache fills enabled, write-through and invalidates enabled

CD = 1, NW = 0

Cache fills are disabled but write-throughs and invalidates are enabled. This mode is the same as if the KEN# pin was strapped HIGH disabling cache fills. Write-throughs and invalidates may still occur to keep the cache valid. This mode is useful if the software must disable the cache for a short period of time, and then re-enable it without flushing the original contents.

CD = 0, NW = 1

Invalid. If CR0 is loaded with this bit configuration, a General Protection fault with error code of 0 will occur.

CD = 0, NW = 0

This is the normal operating mode.

Completely disabling the cache is a two-step process. First, CD and NW must be set to 1, and then the cache must be flushed. If the cache is not flushed, cache hits on reads will still occur and data will be read from the cache.

7.3 Cache Line Fills

Any area of memory can be cached in the Military Intel486 processor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the Military Intel486 processor that a memory address is non-cacheable by returning the KEN# pin inactive during a memory access. (Refer to section 10.2.3, ''Cacheable Cycles.'') Software can prevent certain pages from being cached by setting the PCD bit in the page table entry.

A read request can be generated from program operation or by an instruction pre-fetch. The data will be supplied from the on-chip cache if a cache hit occurs on the read address. If the address is not in the cache, a read request for the data is generated on the external bus.

If the read request is to a cacheable portion of memory, the Military Intel486 processor initiates a cache line fill. During a line fill a 16-byte line is read into the Military Intel486 processor. Cache line fills will only be generated for read misses. Write misses will never cause a line in the internal cache to be allocated. If a cache hit occurs on a write, the line will be updated. Cache line fills can be performed over 8- and 16-bit buses using the dynamic bus sizing feature. Refer to section 10.1.2, ''Dynamic Data Bus Sizing'' for a description of dynamic bus sizing and section 10.2.3, ''Cacheable Cycles'' for further information on cacheable cycles.

7.4 Cache Line Invalidations

The Military Intel486 processor contain both a hardware and software mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the Military Intel486 processor cache contents consistent with external memory.

Refer to section 10.2.8, ''Invalidate Cycles'' for further information on cache line invalidations.

7.5 Cache Replacement

When a line needs to be placed in its internal cache the Military Intel486 processor first checks to see if there is a non-valid line in the set that can be replaced. If all four lines in the set are valid, a pseudo least-recently-used mechanism is used to determine which line should be replaced.

A valid bit is associated with each line in the cache. When a line needs to be placed in a set, the four valid bits are checked to see if there is a non-valid line that can be replaced. If a non-valid line is found, that line is marked for replacement.

The four lines in the set are labeled I0, I1, I2, and I3. The order in which the valid bits are checked during an invalidation is I0, I1, I2 and I3. All valid bits are cleared when the processor is reset or when the cache is flushed.

Replacement in the cache is handled by a pseudo least recently used (LRU) mechanism when all four lines in a set are valid. Three bits, B0, B1 and B2, are defined for each of the 128 sets in the cache. These bits are called the LRU bits. The LRU bits are updated for every hit or replace in the cache.

If the most recent access to the set was to I0 or I1, B0 is set to 1. B0 is set to 0 if the most recent access was to I2 or I3. If the most recent access to I0:I1 was to I0, B1 is set to 1, else B1 is set to 0. If the most recent access to I2:I3 was to I2, B2 is set to 1, else B2 is set to 0.

The pseudo LRU mechanism works in the following manner. When a line must be replaced, the cache will first select which of I0:I1 and I2:I3 was least recently used. Then the cache will determine which of the two lines was least recently used and mark it for replacement. This decision tree is shown in Figure 7-2.

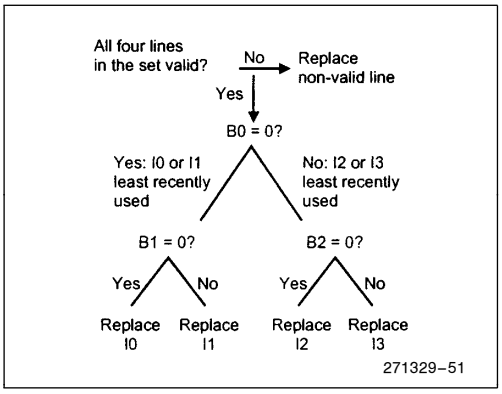


Figure 7-2. On-Chip Cache Replacement Strategy

7.6 Page Cacheability

Two bits for cache control, PWT and PCD, are defined in the page table and page directory entries. The state of these bits are driven out on the PWT and PCD pins during memory access cycles.

The PWT bit controls the write policy for second level caches used with the Military Intel486 processor. Setting PWT = 1 defines a write-through policy for the current page while PWT = 0 defines the possibility of write-back. The state of PWT is ignored internally by the Military Intel486 processor for on-chip cache in write through mode.

The PCD bit controls cacheability on a page by page basis. The PCD bit is internally AND'ed with the KEN# signal to control cacheability on a cycle by cycle basis (see Figure 7-3). PCD = 0 enables caching while PCD = 1 forbids it. Note that cache fills are enabled when PCD = 0 AND KEN# = 0. This logical AND is implemented physically with a NOR gate.

The state of the PCD bit in the page table entry is driven on the PCD pin when a page in external memory is accessed. The state of the PCD pin informs the external system of the cacheability of the requested information. The external system then returns KEN# telling the Military Intel486 processor if the area is cacheable. The Military Intel486 processor initiates a cache line fill if PCD and KEN# indicate that the requested information is cacheable.

The PCD bit is OR'ed with the CD (cache disable) bit in control register 0 to determine the state of the PCD pin. If CD = 1, the Military Intel486 processor forces the PCD pin HIGH. If CD = 0, the PCD pin is driven with the value for the page table entry/directory. (See Figure 7-3.)

The PWT and PCD bits for a bus cycle are obtained from either CR3, the page directory or page table entry. These bits are assumed to be zero during real mode, whenever paging is disabled, or for cycles that bypass paging, (I/O references, interrupt acknowledge and HALT cycles).

When paging is enabled, the bits from the page table entry are cached in the TLB, and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles, PWT and PCD are taken from the page table entry. During TLB refresh cycles where the page table and directory entries are read, the PWT and PCD bits must be obtained elsewhere. During page table updates the bits are obtained from the page directory. When the page directory is updated the bits are obtained from CR3. PCD and PWT bits are initialized to zero at reset, but can be modified by level 0 software.

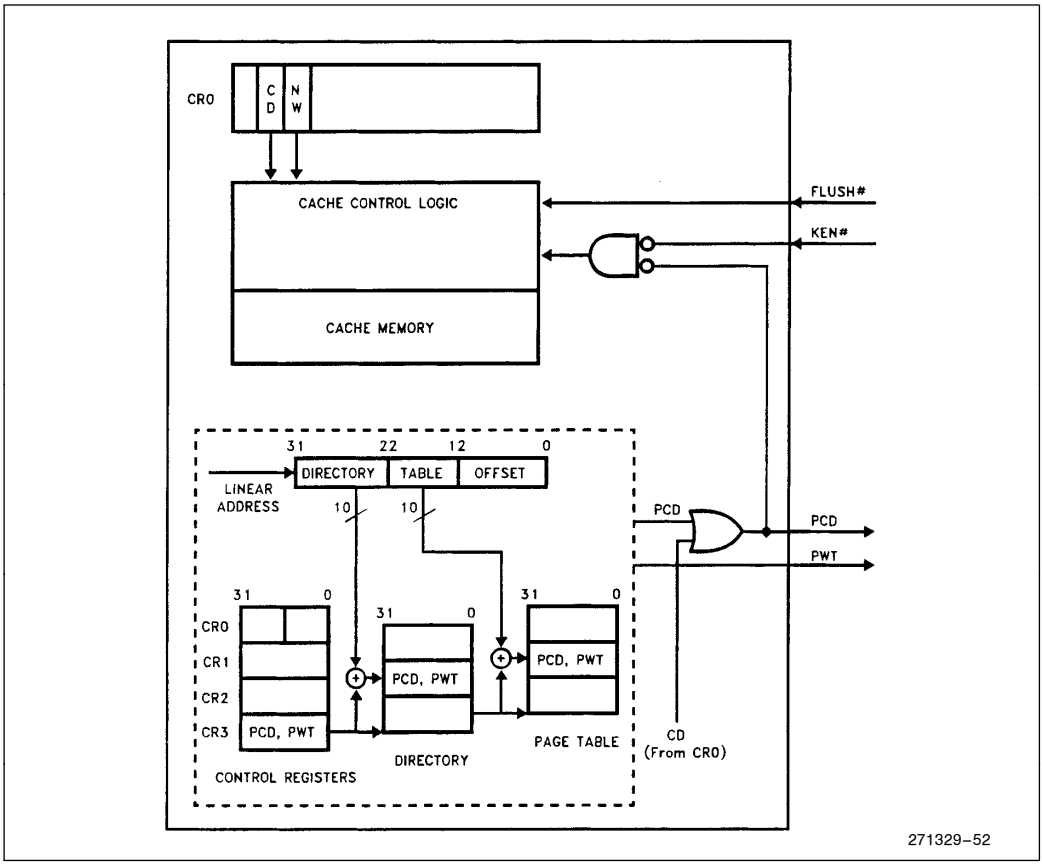


Figure 7-3. Page Cacheability

7.7 Cache Flushing

The on-chip cache can be flushed by external hardware or by software instructions. Flushing the cache clears all valid bits for all lines in the cache. The cache is flushed when external hardware asserts the FLUSH# pin.

The FLUSH# pin needs to be asserted for one clock if driven synchronously or for two clocks if driven asynchronously. FLUSH# is asynchronous, but setup and hold times must be met for recognition in a particular cycle. FLUSH# should be de-asserted before the cache flush is complete. Failure to de-assert the pin will cause execution to stop as the processor will be repeatedly flushing the cache. If external hardware activates flush in response to an I/O write, FLUSH# must be asserted for at least two clocks prior to ready being returned for the I/O write. This ensures that the flush completes before the processor begins execution of the instruction following the OUT instruction.

The instructions INVD and WBINVD cause the on-chip cache to be flushed. External caches connected to the Military Intel486 processor are signaled to flush their contents when these instructions are executed.

WBINVD will also cause an external write-back cache to write back dirty lines before flushing its contents. The external cache is signaled using the bus cycle definition pins and the byte enables (refer to section 9.2.6 ''Bus Cycle Definition'' for the bus cycle definition pins and section 10.2.11 ''Special Bus Cycles'' for special bus cycles). Refer to the *Military Intel486™ Processor Programmers Reference Manual* for detailed instruction definitions.

The results of the INVD and WBINVD instructions are identical for the operation of the non-write-back enhanced Military Intel486 processor on-chip cache because the cache is write-through.

8.0 SYSTEM MANAGEMENT MODE (SMM) ARCHITECTURES

8.1 SMM Overview

The Military Intel486 processor supports four modes: Real, Virtual-86, Protected, and System Management Mode (SMM). As an operating mode, SMM has a distinct processor environment, interface and hardware/software features.

SMM provides system designers with a means of adding new software-controlled features to computer products that operate transparently to the operating system and software applications. SMM is intended for use only by system firmware, not by applications software or general purpose systems software.

The SMM architectural extension consists of the following elements:

1. System Management Interrupt (SMI#) hardware interface.
2. Dedicated and secure memory space (SMRAM) for SMI# handler code and processor state (context) data with a status signal for the system to decode access to that memory space, SMIACT#. (The SMBASE address is relocatable and could also be relocated to non-cacheable address space.)
3. Resume (RSM) instruction, for exiting the System Management Mode.
4. Special Features such as I/O-Restart, for transparent power management of I/O peripherals, and Auto HALT Restart.

8.2 Terminology

The following terms are used throughout the discussion of System Management Mode.

SMM: System Management Mode. This is the operating environment that the processor (system) enters when the System Management Interrupt is being serviced.

SMI#: System Management Interrupt. This is part of the SMM interface. When SMI# is asserted (SMI# pin asserted low) it causes the processor to invoke SMM. **The SMI# pin is the only means of entering SMM.**

SMM handler: System Management Mode handler. This is the code that will be executed when the processor is in SMM. An example application that this code might implement is a power management control or a system control function.

RSM: Resume instruction. This instruction is used by the SMM handler to exit the SMM and return to the interrupted operating system or application process.

SMRAM: This is the physical memory dedicated to SMM. The SMM handler code and related data reside in this memory. This memory is also used by the processor to store its context before executing the SMM handler. The operating system and applications do not have access to this memory space.

SMBASE: Control register that contains the address of the SMRAM space.

Context: This term refers to the processor state. The SMM discussion refers to the context, or processor state, just before the processor invokes SMM. The context normally consists of the processor registers that fully represent the processor state.

Context Switch: A context switch is the process of either saving or restoring the context. The SMM discussion refers to the context switch as the process of saving/restoring the context while invoking/exiting SMM, respectively.

8.3 System Management Interrupt Processing

The system interrupts the normal program execution and invokes SMM by generating a System Management Interrupt (SMI#) to the processor. The processor will service the SMI# by executing the following sequence (see Figure 8-1):

1. The processor asserts the SMIACT# signal, indicating to the system that it should enable the SMRAM.
2. The processor saves its state (context) to SMRAM, starting at default address location 3FFFFH, proceeding downward in a stack-like fashion.
3. The processor switches to the System Management Mode processor environment (a pseudo-real mode).
4. The processor will then jump to the default absolute address of 38000H in SMRAM to execute the SMI# handler. This SMI# handler performs the system management activities.
5. The SMI# handler will then execute the RSM instruction which restores the processors context from SMRAM, de-asserts the SMIACT# signal, and then returns control to the previously interrupted program execution.

NOTE:
The above sequence is valid for the default SMBASE value only. See the following sections for a description of the SMBASE register and SMBASE relocation.

The System Management Interrupt hardware interface consists of the SMI# interrupt request input and the SMIACT# output used by the system to decode the SMRAM.

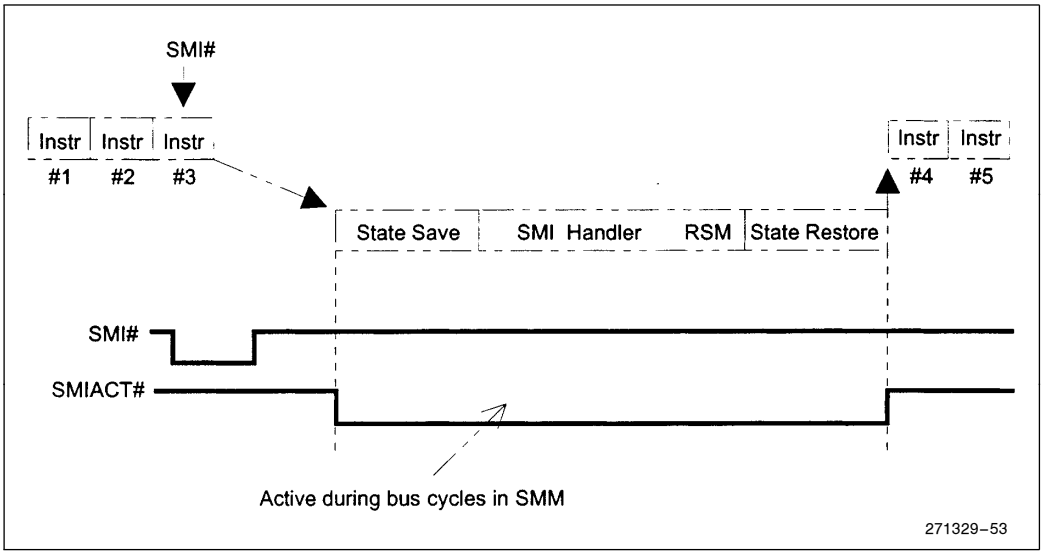


Figure 8-1. Basic SMI# Interrupt Service

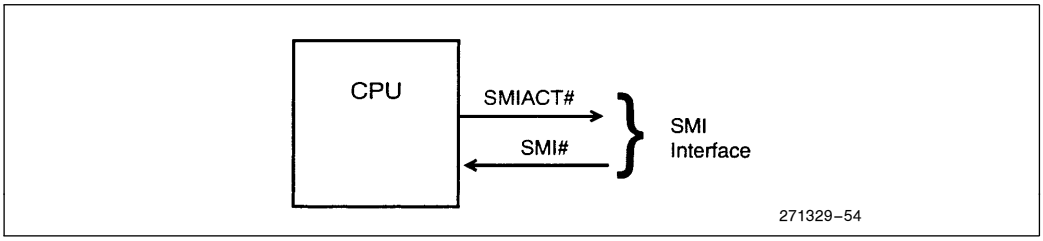


Figure 8-2. Basic SMI# Hardware Interface

8.3.1 SYSTEM MANAGEMENT INTERRUPT (SMI#)

SMI# is a falling-edge triggered, non-maskable interrupt request signal. SMI# is an asynchronous signal, but setup and hold times, t_{20} and t_{21}, must be met in order to guarantee recognition on a specific clock. The SMI# input need not remain active until the interrupt is actually serviced. The SMI# input only needs to remain active for a single clock if the required setup and hold times are met. SMI# will also work correctly if it is held active for an arbitrary number of clocks.

The SMI# input must be held inactive for at least four external clocks after it is asserted to reset the edge triggered logic. A subsequent SMI# might not be recognized if the SMI# input is not held inactive for at least four clocks after being asserted.

SMI#, like NMI, is not affected by the IF bit in the EFLAGS register and is recognized on an instruction boundary. An SMI# will not break locked bus cycles. The SMI# has a higher priority than NMI and is not masked during an NMI. In order for SMI# to be recognized with respect to SRESET, SMI# should not be asserted until two (2) clocks after SRESET becomes inactive.

After the SMI# interrupt is recognized, the SMI# signal will be masked internally until the RSM instruction is executed and the interrupt service routine is complete. Masking the SMI# prevents recursive SMI# calls. SMI# must be de-asserted for at least 4 clocks to reset the edge triggered logic. If another SMI# occurs while the SMI# is masked, the pending SMI# will be recognized and executed on the next instruction boundary after the current SMI# completes. This instruction boundary occurs before execution of the next instruction in the interrupted application code, resulting in back to back SMM handlers. Only one SMI# can be pending while SMI# is masked.

The SMI# signal is synchronized internally and must be asserted at least three (3) CLK periods prior to asserting the RDY# signal in order to guarantee recognition on a specific instruction boundary. This is important for servicing an I/O trap with an SMI# handler. (See Figure 8-3.)

8.3.2 SMI# ACTIVE (SMIACT#)

SMIACT# indicates that the processor is operating in System Management Mode. The processor asserts SMIACT# in response to an SMI# interrupt request on the SMI# pin. SMIACT# is driven active after the processor has completed all pending write cycles (including emptying the write buffers), and before the first access to SMRAM when the processor saves (writes) its state (or context) to SMRAM. SMIACT# remains active until the last access to SMRAM when the processor restores (reads) its state from SMRAM. The SMIACT# signal does not float in response to HOLD. The SMIACT# signal is used by the system logic to decode SMRAM (See Figure 8-2).

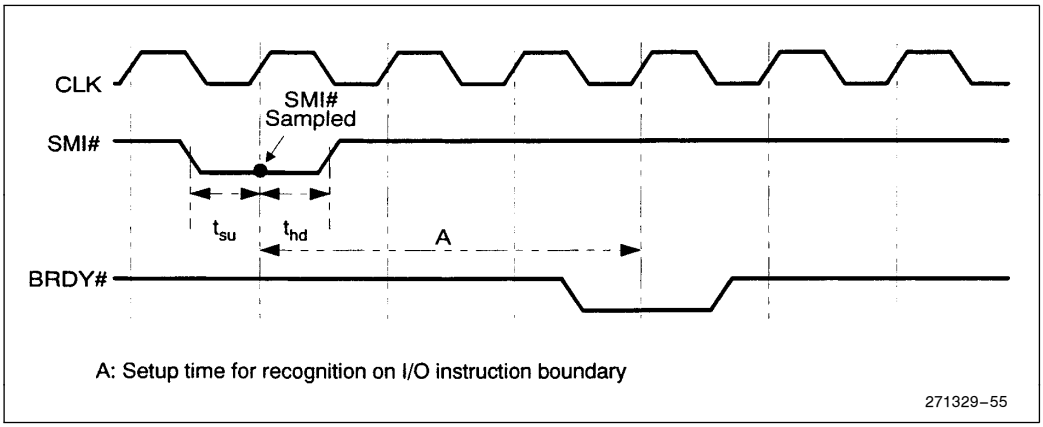


Figure 8-3. SMI# Timing for Servicing an I/O Trap

The number of CLKs required to complete the SMM state save and restore is very dependent on-system memory performance. The values listed in Table 8-1 assume 0 wait-state memory writes (2 CLK cycles), 2-1-1-1 burst read cycles, and 0 wait-state non-burst reads (2 CLK cycles). Additionally, it is assumed that the data read during the SMM state restore sequence is not cacheable.

Figure 8-4 and Table 8-1 can be used for latency calculations. As shown, the minimum time required to enter an SMI# handler routine for the Military Intel486 DX processor (from the completion of the interrupted instruction) is given by:

Latency to beginning of SMI# handler =

A + B + C = 153 CLKs

and the minimum time required to return to the interrupted application (following the final SMM instruction before RSM) is given by:

Latency to continue interrupted application =

E + F + G = 243 CLKs

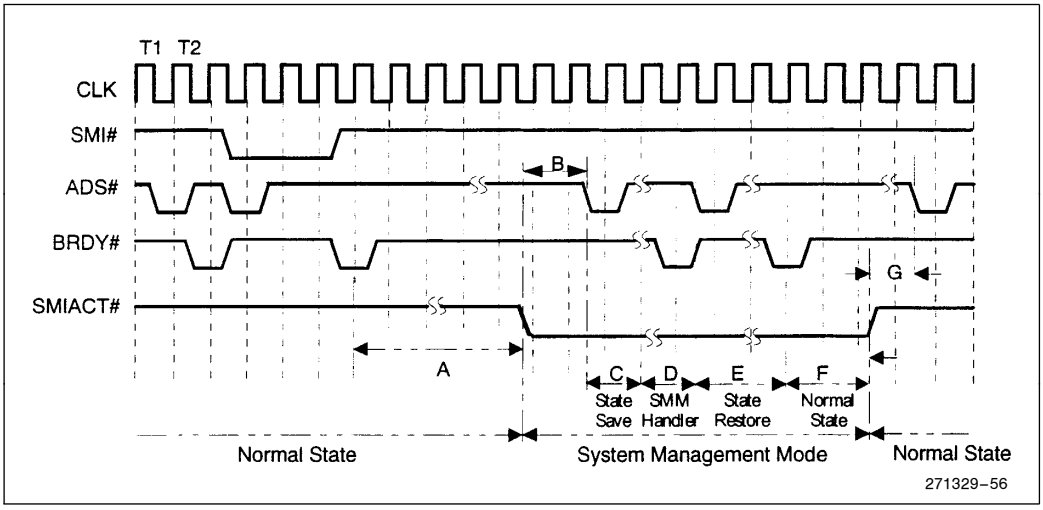


Figure 8-4. Military Intel486™ Processor SMIACT# Timing

Table 8-1. Military Intel486™ Processor SMIACT# Timing

	Military Intel486 DX Processor	IntelDX2™ Processor	IntelDX4™ Processor 3X	IntelDX4 Processor 2X
A:Last RDY# from non-SMM transfer to SMIACT# assertion	2 CLK minimum	1 CLK minimum	1 CLK minimum	1 CLK minimum
B:SMIACT# assertion to first ADS# for SMM state save	40 CLK minimum	20 CLK minimum	13 CLK minimum	20 CLK minimum
C:SMM state save (dependent on memory performance)	Approx. 139 CLKs	Approx. 139 CLKs	Approx. 139 CLKs	Approx. 139 CLKs
D:SMM handler	User determined	User determined	User determined	User determined
E:SMM state restore (dependent on memory performance)	Approx. 236 CLKs	Approx. 236 CLKs	Approx. 236 CLKs	Approx. 236 CLKs
F:Last RDY# from SMM transfer to de-assertion of SMIACT#	4 CLK minimum	2 CLK minimum	1 CLK minimum	1 CLK minimum
G: SMIACT# de-assertion to first non-SMM ADS#	20 CLK minimum	10 CLK minimum	6 CLK minimum	10 CLK minimum

8.3.3 SMRAM

The Military Intel486 processor uses the SMRAM space for state save and state restore operations during an SMI# and RSM. The SMI# handler, which also resides in SMRAM, uses the SMRAM space to store code, data and stacks. In addition, the SMI# handler can use the SMRAM for system management information such as the system configuration, configuration of a powered-down device, and system designer-specific information.

The processor asserts the SMIACT# output to indicate to the memory controller that it is operating in System Management Mode. The system logic should ensure that only the processor has access to this area. Alternate bus masters or DMA devices trying to access the SMRAM space when SMIACT# is active should be directed to system RAM in the respective area.

The system logic is minimally required to decode the physical memory address range from 38000H–3FFFFH as SMRAM area. The processor will save its state to the state save area from 3FFFFH downward to 3FE00H. After saving its state the processor will jump to the address location 38000H to begin executing the SMI# handler. The system logic can choose to decode a larger area of SMRAM as needed. The size of this SMRAM can be between 32 Kbytes and 4 Gbytes.

The system logic should provide a manual method for switching the SMRAM into system memory space when the processor is **not** in SMM. This will enable initialization of the SMRAM space (i.e., loading SMI# handler) before executing the SMI# handler during SMM. (See Figure 8-5.)

8.3.3.1 SMRAM State Save Map

When the SMI# is recognized on an instruction boundary, the processor core first sets the SMIACT# signal LOW indicating to the system logic that accesses are now being made to the system-defined SMRAM areas. The processor then writes its state to the state save area in the SMRAM. The state save area starts at CS Base + [8000H + 7FFFH]. The default CS Base is 30000H, therefore the default state save area is at 3FFFFH. In this case, the CS Base can also be referred to as the SMBASE.

If SMBASE Relocation is enabled, then the SMRAM addresses can change. The following formula is used to determine the relocated addresses where the context is saved. The context will reside at CS Base + [8000H + Register Offset], where the default initial CS Base is 30000H and the Register Offset is listed in the SMRAM state save map (Table 8-2). Reserved spaces will be used to accommodate new registers in future processors. The state save area starts at 7FFFH and continues downward in a stack-like fashion.

Some of the registers in the SMRAM state save area may be read and changed by the SMI# handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). The values stored in the areas marked reserved may change in future processors. An SMM handler should not rely on any values stored in an area that is marked as reserved.

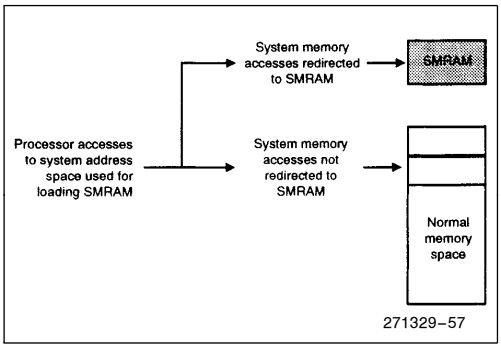


Figure 8-5. Redirecting System Memory Addresses to SMRAM

Table 8-2. SMRAM State Save Map

Register Offset	Register	Writeable?
7FFC	CR0	NO
7FF8	CR3	NO
7FF4	EFLAGS	YES
7FF0	EIP	YES
7FEC	EDI	YES
7FE8	ESI	YES
7FE4	EBP	YES
7FE0	ESP	YES
7FDC	EBX	YES
7FD8	EDX	YES
7FD4	ECX	YES
7FD0	EAX	YES
7FCC	DR6	NO
7FC8	DR7	NO
7FC4	TR*	NO
7FC0	LDTR*	NO
7FBC	GS*	NO
7FB8	FS*	NO
7FB4	DS*	NO
7FB0	SS*	NO
7FAC	CS*	NO
7FA8	ES*	NO
7FA7–7F98	Reserved	NO
7F94	IDT Base	NO
7F93–7F8C	Reserved	NO
7F88	GDT Base	NO
7F87–7F04	Reserved	NO
7F02	Auto HALT Restart Slot (Word)	YES
7F00	I/O Trap Restart Slot (Word)	YES
7EFC	SMM Revision Identifier (Dword)	NO
7EF8	SMBASE Slot (Dword)	YES
7EF7–7E00	Reserved	NO

NOTES:
*Upper two bytes are reserved.
Modifying a value that is marked as not writeable will result in unpredictable behavior.
Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address and the high-order byte in the high address.

The following registers are saved and restored (in areas of the state save that are marked reserved), but are not visible to the system software programmer:

CR1, CR2 and CR4, hidden descriptor registers for CS, DS, ES, FS, GS, and SS.

If an SMI# request is issued for the purpose of powering down the processor, the values of all reserved locations in the SMM state save must be saved to non-volatile memory.

The following registers are not automatically saved and restored by SMI# and RSM:

DR5–DR0, TR7–TR3, FPU registers: STn, FCS, FSW, tag word, FP instruction pointer, FP opcode, and operand pointer.

For all SMI# requests except for suspend/resume, these registers do not have to be saved because their contents will not change. However, during a power down suspend/resume, a resume reset will clear these registers back to their default values. In this case, the suspend SMI# handler should read these registers directly to save them and restore them during the power up resume. Anytime the SMI# handler changes these registers in the processor, it must also save and restore them.

8.3.4 EXIT FROM SMM

The **RSM** instruction is only available to the SMI# handler. The opcode of the instruction is 0FAAH. Execution of this instruction while the processor is executing outside of SMM will cause an invalid opcode error. The last instruction of the SMI# handler will be the RSM instruction.

The RSM instruction restores the state save image from SMRAM back to the processor, then returns control back to the interrupted program execution. There are three SMM features that can be enabled by writing to control ''slots'' in the SMRAM state save area.

Auto HALT Restart. It is possible for the SMI# request to interrupt the HALT state. The SMI# handler can tell the RSM instruction to return control to the HALT instruction **or** to return control to the instruction following the HALT instruction by appropriately setting the Auto HALT Restart slot. The default operation is to restart the HALT instruction.

I/O Trap Restart. If the SMI# interrupt was generated on an I/O access to a powered-down device, the SMI# handler can tell the RSM instruction to re-execute that I/O instruction by setting the I/O Trap Restart slot.

SMBASE Relocation. The system can relocate the SMRAM by setting the SMBASE Relocation slot in the state save area. The RSM instruction will set the SMBASE in the processor based on the value in the SMBASE relocation slot. The SMBASE must be 32K aligned.

For further details on these SMM features, see section 8.5.

If the processor detects invalid state information, it enters the shutdown state. This happens only in the following situations:

- The value stored in the SMBASE slot is not a 32-Kbyte-aligned address.
- A reserved bit of CR4 is set to 1.
- A combination of bits in CR0 is illegal; namely, (PG = 1 and PE = 0) or (NW = 1 and CD = 0).

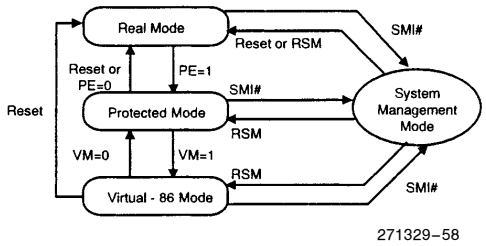
In shutdown mode, the processor stops executing instructions until an NMI interrupt is received or reset initialization is invoked. The processor generates a special bus cycle to indicate it has entered shutdown mode.

NOTE:
INTR and SMI# will also bring the processor out of a shutdown that is encountered due to invalid state information from SMM execution. Make sure that INTR and SMI# are not asserted if SMM routines are written such that a shutdown occurs.

8.4 System Management Mode Programming Model

8.4.1 ENTERING SYSTEM MANAGEMENT MODE

SMM is one of the major operating modes, on a level with Protected mode, Real address mode or virtual-86 mode. Figure 8-6 shows how the processor can enter SMM from any of the three modes and then return.



NOTE:
Reset could occur by asserting the RESET or SRESET pin.

Figure 8-6. Transition to and from System Management Mode

The external signal SMI# causes the processor to switch to SMM. The RSM instruction exits SMM. SMM is transparent to applications programs and operating systems because of the following:

- The only way to enter SMM is via a type of non-maskable interrupt triggered by an external signal.
- The processor begins executing SMM code from a separate address space, referred to earlier as system management RAM (SMRAM).
- Upon entry into SMM, the processor saves the register state of the interrupted program in a part of SMRAM called the SMM context save space.
- All interrupts normally handled by the operating system or by applications are disabled upon entry into SMM
- A special instruction, RSM, restores processor registers from the SMM context save space and returns control to the interrupted program.

SMM is similar to Real address mode in that there are no privilege levels or address mapping. An SMM program can execute all I/O and other system instructions and can address up to four Gbytes of memory.

8.4.2 PROCESSOR ENVIRONMENT

When an SMI# signal is recognized on an instruction execution boundary, the processor waits for all stores to complete, including emptying of the write buffers. The final write cycle is complete when the system returns RDY# or BRDY#. The processor then drives SMIACT# active, saves its register state to SMRAM space, and begins to execute the SMM handler.

SMI# has greater priority than debug exceptions and external interrupts. This means that if more than one of these conditions occur at an instruction boundary, only the SMI# processing occurs, not a debug exception or external interrupt. Subsequent SMI# requests are not acknowledged while the processor is in SMM. The first SMI# interrupt request that occurs while the processor is in SMM is latched, and serviced when the processor exits SMM with the RSM instruction. Only one SMI# will be latched by the processor while it is in SMM.

When the processor invokes SMM, the processor core registers are initialized as shown in Table 8-3.

Table 8-3. SMM Initial Processor Core Register Settings

Register	Contents
General Purpose Registers	Unpredictable
EFLAGS	00000002H
EIP	00008000H
CS Selector	3000H
CS Base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	000000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFFH
CR0	Bits 0,2,3 & 31 cleared (PE, EM, TS & PG); others are unmodified
DR6	Unpredictable
DR7	00000000H

The following is a summary of the key features in the SMM environment:

1. Real mode style address calculation
2. 4-Gbyte limit checking
3. IF flag is cleared
4. NMI is disabled
5. TF flag in EFLAGS is cleared; single step traps are disabled
6. DR7 is cleared, except for bits 12 and 13; debug traps are disabled.
7. The RSM instruction no longer generates an invalid opcode error
8. Default 16-bit opcode, register and stack use.

All bus arbitration (HOLD, AHOLD, BOFF#) inputs and bus sizing (BS8#, BS16#) inputs operate normally while the processor is in SMM.

8.4.3 EXECUTING SYSTEM MANAGEMENT MODE HANDLER

The processor begins execution of the SMM handler at offset 8000H in the CS segment. The CS Base is initially 30000H. However, the CS Base can be changed by using the SMM Base relocation feature.

When the SMM handler is invoked, the processors PE and PG bits in CR0 are reset to 0. The processor is in an environment similar to Real mode, but without the 64-Kbyte limit checking. However, the default operand size and the default address size are set to 16 bits.

The EM bit is cleared so that no exceptions are generated. (If the SMM was entered from Protected mode, the Real mode interrupt and exception support is not available.) The SMI# handler should not use floating point unit instructions until the FPU is properly detected (within the SMI# handler) and the exception support is initialized.

Because the segment bases (other than CS) are cleared to 0 and the segment limits are set to 4 Gbytes, the address space may be treated as a single flat 4-Gbyte linear space that is unsegmented. The processor is still in Real mode and when a segment selector is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base cache. The limits and attributes are not modified.

In SMM, the processor can access or jump anywhere within the 4-Gbyte logical address space. The processor can also indirectly access or perform a near jump anywhere within the 4-Gbyte logical address space.

8.4.3.1 Exceptions and Interrupts within System Management Mode

When the processor enters SMM, it disables INTR interrupts, debug and single-step traps by clearing the EFLAGS, DR6 and DR7 registers. This is done to prevent a debug application from accidentally breaking into an SMM handler. This is necessary because the SMM handler operates from a distinct address space (SMRAM), and hence, the debug trap will not represent the normal system memory space.

If an SMM handler wishes to use the debug trap feature of the processor to debug SMM handler code, it must first ensure that an SMM compliant debug handler is available. The SMM handler must also ensure DR0–DR3 is saved to be restored later. The debug registers DR0–DR3 and DR7 must then be initialized with the appropriate values.

If the processor wishes to use the single step feature of the processor, it must ensure that an SMM compliant single step handler is available and then set the trap flag in the EFLAGS register.

If the system design requires the processor to respond to hardware INTR requests while in SMM, it must ensure that an SMM compliant interrupt handler is available and then set the interrupt flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked upon entry to SMM, and the system software designer must provide an SMM compliant interrupt handler before attempting to execute any software interrupt instructions. Note that in SMM mode, the interrupt vector table has the same properties and location as the Real mode vector table.

NMI interrupts are blocked upon entry to the SMM handler. If an NMI request occurs during the SMM handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMM handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence.

Although NMI requests are blocked when the processor enters SMM, they may be enabled through software by executing an IRET instruction. If the SMM handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same "real mod" manner in which they are handled outside of SMM.

8.5 SMM Features

8.5.1 SMM REVISION IDENTIFIER

The SMM revision identifier is used to indicate the version of SMM and the SMM extensions that are supported by the processor. The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at Register Offset 7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture. The upper word of the SMM revision identifier refers to the extensions available. (See Figure 8-7.)

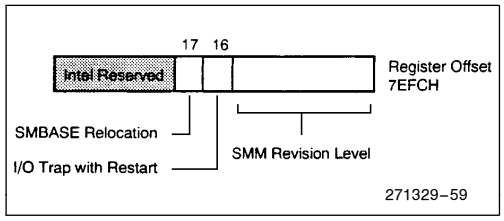


Figure 8-7. SMM Revision Identifier

Table 8-4. Bit Values for SMM Revision Identifier

Bits	Value	Comments
16	0	Processor does not support I/O Trap Restart
16	1	Processor supports I/O Trap Restart
17	0	Processor does not support SMBASE relocation
17	1	Processor supports SMBASE relocation

Bit 16 of the SMM revision identifier is used to indicate to the SMM handler that this processor supports the SMM I/O trap extension. If this bit is high, then this processor supports the SMM I/O trap extension. If this bit is low, then this processor does not support I/O trapping using the I/O trap slot mechanism. (See Table 8-4.)

Bit 17 of this slot indicates whether the processor supports relocation of the SMM jump vector and the SMRAM base address. (See Table 8-4.)

The Military Intel486 processor supports both the I/O Trap Restart and the SMBASE relocation features.

8.5.2 AUTO HALT RESTART

The Auto HALT Restart slot at register offset (word location) 7F02H in SMRAM indicates to the SMM handler that the SMI# interrupted the processor during a HALT state (bit 0 of slot 7F02H is set to 1 if the previous instruction was a HALT). If the SMI# did not interrupt the processor in a HALT state, then the SMI# microcode will set bit 0 of the Auto HALT Restart slot to a value of 0. If the previous instruction was a HALT, the SMM handler can choose to either set or reset bit 0. If this bit is set to 1, the RSM micro code execution will force the processor to re-enter the HALT state. If this bit is set to 0 when the RSM instruction is executed, the processor will continue execution with the instruction just after the interrupted HALT instruction. Note that if the interrupted instruction was not a HALT instruction (bit 0 is set to 0 in the Auto HALT Restart slot upon SMM entry), setting bit 0 to 1 will cause unpredictable behavior when the RSM instruction is executed. (See Figure 8-8 and Table 8-5.)

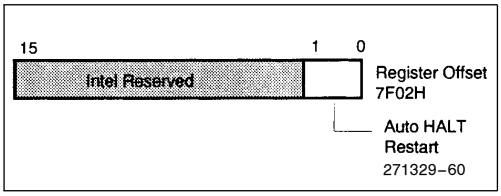


Figure 8-8. Auto HALT Restart

Table 8-5. Bit Values for Auto HALT Restart

Value of Bit 0 at Entry	Value of Bit 0 at Exit	Comments
0	0	Returns to next instruction in interrupted program.
0	1	Unpredictable
1	0	Returns to next instruction after HALT
1	1	Returns to HALT state

If the HALT instruction is restarted, the processor will generate a memory access to fetch the HALT instruction (if it is not in the internal cache), and execute a HALT bus cycle.

8.5.3 I/O INSTRUCTION RESTART

The I/O instruction restart slot (register offset 7F00H in SMRAM) gives the SMM handler the option of causing the RSM instruction to automatically re-execute the interrupted I/O instruction. When the RSM instruction is executed, if the I/O instruction restart slot contains the value 0FFH, then the processor will automatically re-execute the I/O instruction that the SMI# trapped. If the I/O instruction restart slot contains the value 00H when the RSM instruction is executed, then the processor will not re-execute the I/O instruction. The processor automatically initializes the I/O instruction restart slot to 00H during SMM entry. The I/O instruction restart slot should be written only when the processor has generated an SMI# on an I/O instruction boundary. Processor operation is unpredictable when the I/O instruction restart slot is set when the processor is servicing an SMI# that originated on a non-I/O instruction boundary. (See Figure 8-9 and Table 8-6.)

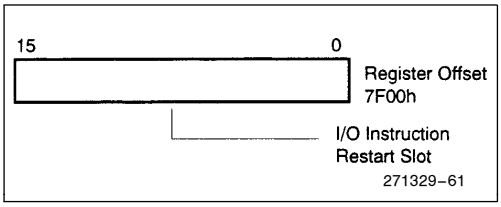


Figure 8-9. I/O Instruction Restart

Table 8-6 I/O Instruction Restart Value

Value at Entry	Value at Exit	Comments
00H	00H	Do not restart trapped I/O instruction
00H	0FFH	Restart trapped I/O instruction

If the system executes back-to-back SMI# requests, the second SMM handler must not set the I/O instruction restart slot (see section 8.6.6 "Nested SMI#s and I/O Restart").

8.5.4 SMM BASE RELOCATION

The Military Intel486 processor provides a control register, SMBASE. The address space used as SMRAM can be modified by changing the SMBASE register before exiting an SMI# handler routine. SMBASE can be changed to any 32K aligned value (values that are not 32K aligned will cause the processor to enter the shutdown state when executing the RSM instruction). SMBASE is set to the default value of 30000H on RESET, but is not changed on SRESET. If the SMBASE register is changed during an SMM handler, all subsequent SMI# requests will initiate a state save at the new SMBASE. (See Figure 8-10.)

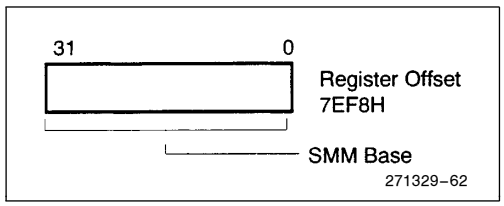


Figure 8-10. SMM Base Location

The SMBASE slot in the SMM state save area is a feature used to indicate and change the SMI# jump vector location and the SMRAM save area. When bit 17 of the SMM Revision Identifier is set then this feature exists and the SMRAM base and consequently the jump vector are as indicated by the SMM Base slot. During the execution of the RSM instruction, the processor will read this slot and initialize the processor to use the new SMBASE during the next SMI#. During an SMI#, the processor will do its context save to the new SMRAM area pointed to by the SMBASE, store the current SMBASE in the SMM Base slot (offset 7EF8H), and then start execution of the new jump vector based on the current SMBASE.

The SMBASE must be a 32-Kbyte aligned, 32-bit integer that indicates a base address for the SMRAM context save area and the SMI# jump vector. For example when the processor first powers up, the minimum SMRAM area is from 38000H–3FFFFH. The default SMBASE is 30000H. Hence the starting address of the jump vector is calculated by:

SMBASE + 8000H

While the starting address for the SMRAM state save area is calculated by:

SMM Base + [8000H + 7FFFH]

Hence, when this feature is enabled, the SMRAM register map is addressed according to the above formulas. (See Figure 8-11.)

To change the SMRAM base address and SMM jump vector location, the SMM handler should modify the SMBASE slot. Upon executing an RSM instruction, the processor will read the SMBASE slot and store it internally. Upon recognition of the next SMI# request, the processor will use the new SMBASE slot for the SMRAM dump and SMI# jump vector.

If the modified SMBASE slot does not contain a 32-Kbyte aligned value, the RSM microcode will cause the processor to enter the shutdown state.

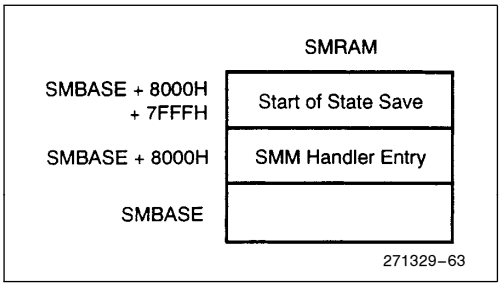


Figure 8-11. SMRAM Usage

8.6 SMM System Design Considerations

8.6.1 SMRAM INTERFACE

The hardware designed to control the SMRAM space must follow these guidelines:

1. A provision should be made to allow for initialization of SMRAM space during system boot up. This initialization of SMRAM space must happen before the first occurrence of an SMI# interrupt. Initializing the SMRAM space must include installation of an SMM handler, and may include installation of related data structures necessary for particular SMM applications. The memory controller providing the interface to the SMRAM should provide a means for the initialization code to manually open the SMRAM space.
2. A minimum initial SMRAM address space of 38000H-3FFFFH should be decoded by the memory controller.
3. Alternate bus masters (such as DMA controllers) should not be allowed to access SMRAM space. Only the processor, either through SMI# or during initialization, should be allowed access to SMRAM.
4. In order to implement a zero-volt suspend function, the system must have access to all of normal system memory from within an SMM handler routine. If the SMRAM is going to overlay normal system memory, there must be a method of accessing any system memory that is located underneath SMRAM.

There are two potential schemes for locating the SMRAM, either overlaid to an address space on top of normal system memory, or placed in a distinct address space. (See Figure 8-12.) When SMRAM is overlaid on the top of normal system memory, the processor output signal SMIACT# must be used to distinguish SMRAM from main system memory. Additionally, if the overlaid normal memory is cacheable, both the processor internal cache and any second level caches must be empty before the first read of an SMM handler routine. If the SMM memory is cacheable, the caches must be empty before the first read of normal memory following an SMM handler routine. This is done by flushing the caches, and is required to maintain cache coherency. When the default SMRAM location is used, SMRAM is overlaid on top of system main memory (at 38000H through 3FFFFH).

If SMRAM is located in its own distinct memory space, which can be completely decoded with only the processor address signals, it is said to be non-overlaid. In this case, there are no new requirements for maintaining cache coherency.

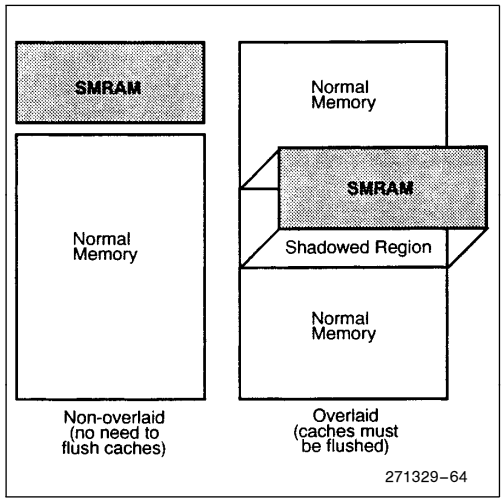


Figure 8-12. SMRAM Location

8.6.2 CACHE FLUSHES

The processor does not unconditionally flush its cache before entering SMM (this option is left to the system designer). If SMRAM is shadowed in a cacheable memory area that is visible to the application or operating system, it is necessary for the system to empty both the processor cache and any second level cache before entering SMM. That is, if SMRAM is in the same physical address location as the normal cacheable memory space, then an SMM read may hit the cache which would contain normal memory space code/data. If the SMM memory is cacheable, the normal read cycles after SMM may hit the cache, which may contain SMM code/data. In this case the cache should be empty before the first memory read cycle during SMM and before the first normal cycle after exiting SMM. (See Figure 8-13.)

The FLUSH# and KEN# signals can be used to ensure cache coherency when switching between normal and SMM modes. Cache flushing during SMM entry is accomplished by asserting the FLUSH# pin when SMI# is driven active. Cache flushing during SMM exit is accomplished by asserting the FLUSH# pin after the SMIACT# pin is deasserted (within 1 CLK). To guarantee this behavior, the constraints on setup and hold timings on the interaction of FLUSH# and SMIACT# as specified for a processor should be followed.

If the SMRAM area is overlaid over normal memory and if the system designer does not want to flush the caches upon leaving SMM then references to the SMRAM area should not be cached. It is the obligation of the system designer to ensure that the KEN# pin is sampled inactive during all references to the SMRAM area. Figures 8-14 and 8-15 illustrate a cached and non-cached SMM using FLUSH# and KEN#.

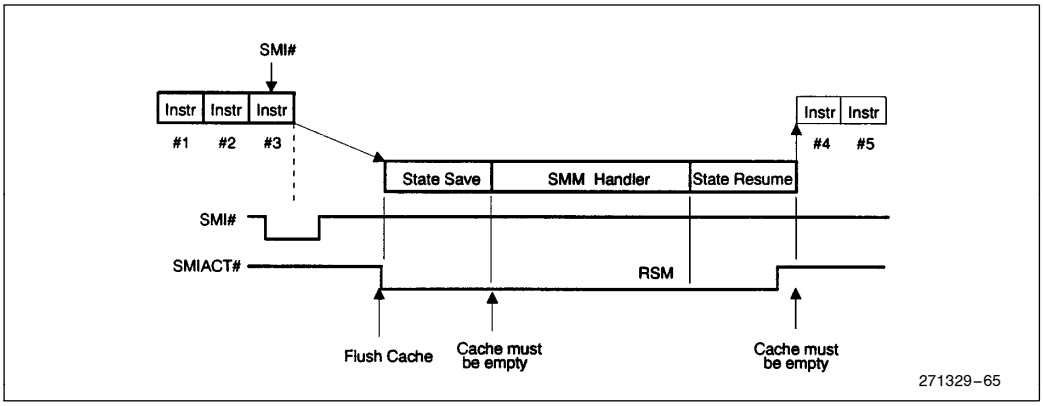


Figure 8-13. FLUSH# Mechanism during SMM

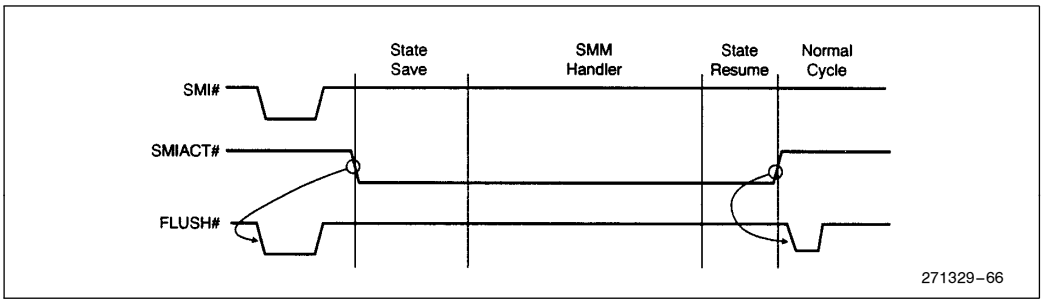


Figure 8-14. Cached SMM

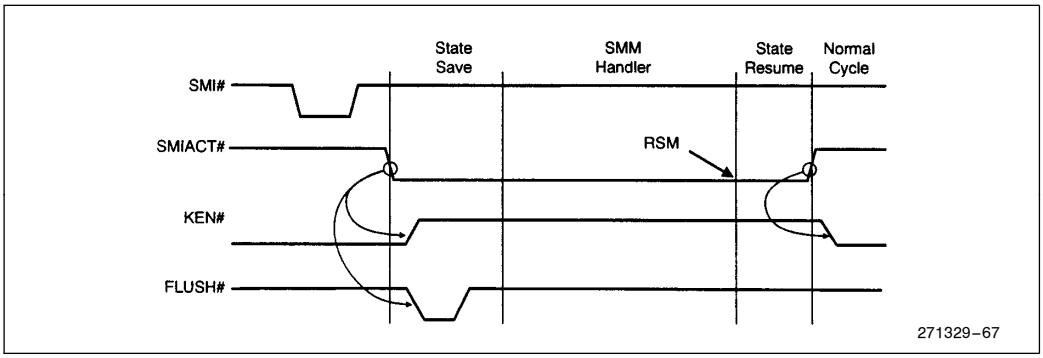


Figure 8-15. Non-Cached SMM

8.6.3 A20M# PIN AND SMBASE RELOCATION

Systems based on a PC-compatible architecture contain a feature that enables the processor address bit A20 to be forced to 0. This limits physical memory to a maximum of 1 Mbyte, and is provided to ensure compatibility with those programs that relied on the physical address wrap around functionality of the 8088 processor. The A20M# pin on Military Intel486 processors provides this function. When A20M# is active, all external bus cycles will drive A20M# low, and all internal cache accesses will be performed with A20M# low.

The A20M# pin is recognized while the processor is in SMM. The functionality of the A20M# input must be recognized in the following two instances:

1. If the SMM handler needs to access system memory space above 1 Mbyte (for example, when saving memory to disk for a zero-volt suspend), the A20M# pin must be de-asserted before the memory above 1 Mbyte is addressed.
2. If SMRAM has been relocated to address space above 1 Mbyte, and A20M# is active upon entering SMM, the processor will attempt to access SMRAM at the relocated address, but with A20 low. This could cause the system to crash, because there would be no valid SMM interrupt handler at the accessed location.

In order to account for the above two situations, the system designer must ensure that A20M# is de-asserted on entry to SMM. A20M# must be driven inactive before the first cycle of the SMM state save, and must be returned to its original level after the last cycle of the SMM state restore. This can be done by blocking the assertion of A20M# whenever SMIACT# is active.

8.6.4 PROCESSOR RESET DURING SMM

The system designer should take into account the following restrictions while implementing the processor RESET logic.

1. When running software written for the 80286 processor a processor SRESET is used to switch the processor from Protected mode to Real mode. Note that SRESET has a higher interrupt priority than SMIACT#. When the processor is in SMM, the SRESET to the processor during SMM should be blocked until the processor exits SMM. SRESET must be blocked beginning from the time when SMI# is driven active and ending at least 20 CLK cycles after SMIACT# is de-asserted. Be careful not to block the global system RESET, which may be necessary to recover from a system crash.
2. During execution of the RSM instruction to exit SMM, there is a small time window between the de-assertion of SMIACT# and the completion of the RSM microcode. If SRESET is asserted during this window, it is possible that the SMRAM space will be violated. The system designer must guarantee that SRESET is blocked until at least 20 processor clock cycles after SMIACT# has been driven inactive.
3. Any request for a processor SRESET for the purpose of switching the processor from Protected mode to Real mode must be acknowledged after the processor has exited SMM. In order to maintain software transparency, the system logic must latch any SRESET signals that are blocked during SMM.

8.6.5 SMM AND SECOND LEVEL WRITE BUFFERS

Before a Military Intel486 processor enters SMM, it empties its internal write buffers. This is necessary so that the data in the write buffers is written to normal memory space, not SMM space. Once the processor is ready to begin writing an SMM state save to SMRAM, it asserts the SMIACT# signal. SMIACT# may be driven active by the processor before the system memory controller has had an opportunity to empty the second level write buffers.

To prevent the data from these second level write buffers from being written to the wrong location, the system memory controller needs to direct the memory write cycles to either SMM space or normal memory space. This can be accomplished by saving the status of SMIACT# along with the address for each word in the write buffers.

8.6.6 NESTED SMI#s AND I/O RESTART

Special care must be taken when executing an SMM handler for the purpose of restarting an I/O instruction. When the processor executes a RSM instruction with the I/O restart slot set, the restored EIP is modified to point to the instruction immediately preceding the SMI# request, so that the I/O instruction

can be re-executed. If a new SMI# request is received while the processor is executing an SMM handler, the processor will service this SMI# request before restarting the original I/O instruction. If the I/O restart slot is set when the processor executes the RSM instruction for the second SMM handler, the RSM microcode will decrement the restored EIP again. EIP now points to an address different from the originally interrupted instruction, and the processor will begin execution of the interrupted application code at an incorrect entry point.

To prevent this from occurring, the SMM handler routine must not set the I/O restart slot during the second of two consecutive SMM handlers.

8.7 SMM Software Considerations

8.7.1 SMM CODE CONSIDERATIONS

The default operand size and the default address size are 16 bits; however, operand-size override and address-size override prefixes can be used as needed to directly access data anywhere within the 4-Gbyte logical address space.

With operand-size override prefixes, the SMM handler can use jumps, calls, and returns, to transfer control to any location within the 4-Gbyte space. Note, however, the following restrictions:

- Any control transfer that does not have an operand-size override prefix truncates EIP to 16 low-order bits.
- Due to the Real mode style of base-address formation, a far jump or call cannot transfer control to a segment with a base address of more than 20 bits (one megabyte).

8.7.2 EXCEPTION HANDLING

Upon entry into SMM, external interrupts that require handlers are disabled (the IF bit in the EFLAGS is cleared). This is necessary because, while the processor is in SMM, it is running in a separate memory space. Consequently the vectors stored in the interrupt descriptor table (IDT) for the prior mode are not applicable. Before allowing exception handling (or software interrupts), the SMM program must initialize new interrupt and exception vectors. The interrupt vector table for SMM has the same format as for Real mode. Until the interrupt vector table is correctly initialized, the SMM handler must not generate an exception (or software interrupt). Even though hardware interrupts are disabled, exceptions and software interrupts can still occur. Only a correctly written SMM handler can prevent internal exceptions. When new exception vectors are initialized, internal exceptions can be serviced. The following are the restrictions:

1. Due to the Real mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more that 20 bits.
2. An interrupt or exception cannot transfer control to a segment offset of more than 16 bits (64 Kbytes).
3. If exceptions or interrupts are allowed to occur, only the low order 16 bits of the return address (EIP) are pushed onto the stack. If the offset of the interrupted procedure is greater than 64 Kbytes, it is not possible for the interrupt/exception handler to return control to that procedure. (One work-around could be to perform software adjustment of the return address on the stack.)
4. The SMBASE Relocation feature affects the way the processor will return from an interrupt or exception during an SMI# handler.

8.7.3 HALT DURING SMM

HALT should not be executed during SMM, unless interrupts have been enabled (see section 8.7.2. ‘‘Exception Handling’’). Interrupts are disabled in SMM and INTR, NMI, and SMI# are the only events that take the processor out of HALT.

8.7.4 RELOCATING SMRAM TO AN ADDRESS ABOVE ONE MEGABYTE

Within SMM (or Real mode), the segment base registers can only be updated by changing the segment register. The segment registers contain only 16 bits, which allows only 20 bits to be used for a segment base address (the segment register is shifted left four bits to determine the segment base address). If SMRAM is relocated to an address above one megabyte, the segment registers can no longer be initialized to point to SMRAM.

These areas can still be accessed by using address override prefixes to generate an offset to the correct address. For example, if the SMBASE has been relocated immediately below 16M, the DS and ES registers are still initialized to 0000 0000H. We can still access data in SMRAM by using 32-bit displacement registers:

```
mov    esi, 00FFxxxxH      ;64K segment
                           ;immediately
                           ;below 16M
mov    ax,ds:[esi]
```

9.0 HARDWARE INTERFACE

9.1 Introduction

The Military Intel486 processor has separate parallel buses for data and addresses. The bidirectional data bus is 32 bits in width. The address bus consists of two components: 30 address lines (A2–A31) and 4-byte enable lines (BE0#–BE3#). The address lines form the upper 30 bits of the address and the byte enables select individual bytes within a 4-byte location. The address lines are bidirectional for use in cache line invalidations. (See Figure 9-1.)

The Military Intel486 processor's burst bus mechanism enables high-speed cache fills from external memory. Burst cycles can strobe data into the processor at a rate of one item every clock. Non-burst cycles have a maximum rate of one item every two clocks. Burst cycles are not limited to cache fills: all read bus cycles requiring more than a single data cycle can be bursted.

During bus hold, the Military Intel486 processor relinquishes control of the local bus by floating its address, data and control buses. The Military Intel486 processor has an address hold feature in addition to bus hold. During address hold, only the address bus is floated, the data and control buses can remain active. Address hold is used for cache line invalidations.

The Military Intel486 supports the IEEE 1149.1 boundary scan.

This section provides a brief description of the Military Intel486 processor input and output signals arranged by functional groups. The # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When a # is not present after the signal name, the signal is active at high voltage level. The term ''ready'' is used to indicate that the cycle is terminated with RDY# or BRDY#.

This section and section 10, ''Bus Operation,'' describe bus cycles and data cycles. A bus cycle is at least two-clocks long and begins with ADS# active in the first clock and RDY# and/or BRDY# active in the last clock. Data is transferred to or from the Military Intel486 processor during a data cycle. A bus cycle contains one or more data cycles.

9.2 Signal Descriptions

9.2.1 CLOCK (CLK)

CLK provides the fundamental timing and the internal operating frequency for the Military Intel486 processor. All external timing parameters are specified with respect to the **rising edge** of CLK.

The Military Intel486 processor can operate over a wide frequency range, however the CLK frequency cannot change rapidly while RESET is inactive. The CLK frequency must be stable for proper chip operation because a single edge of CLK is used internally to generate two phases. CLK only needs TTL levels for proper operation. Figure 9-2 illustrates the CLK waveform.

9.2.2 INTELDX4 PROCESSOR CLOCK MULTIPLIER SELECTABLE INPUT (CLKMUL)

The IntelDX4 processor differs from the IntelDX2 processor in that it provides for two internal clock multiplier ratios: speed doubled mode and speed tripled mode. Speed doubled mode is identical to the IntelDX2 processor mode of operation where the internal core is operating at twice the external bus frequency. Selecting speed tripled mode causes the internal core frequency to operate at three times the external bus frequency. The IntelDX4 processor determines the desired clock multiplier ratio by sampling the status of the CLKMUL input during cold (power on) processor resets. **The clock multiplier ratio cannot be changed during warm resets. Also, SRESET cannot be used to select the clock multiplier ratio.**

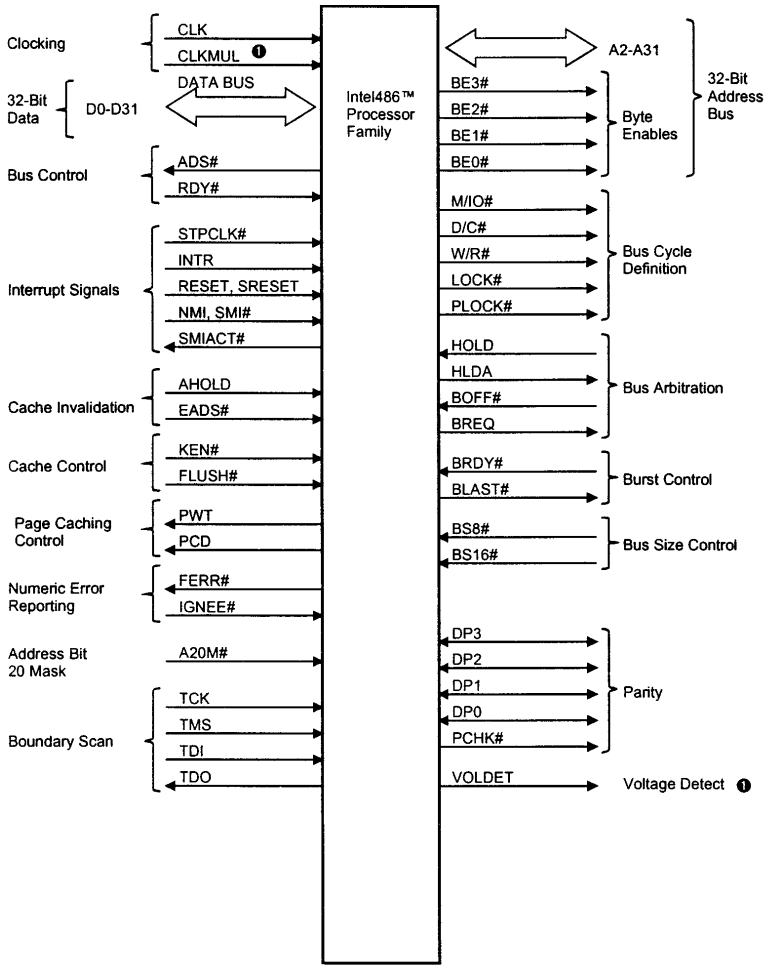
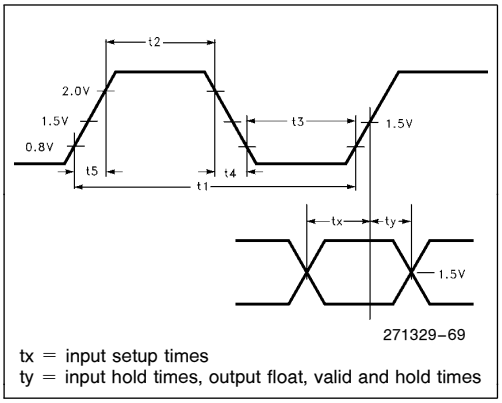


Figure 9-1. Functional Signal Groupings



tx = input setup times
ty = input hold times, output float, valid and hold times

Figure 9-2. CLK Waveform

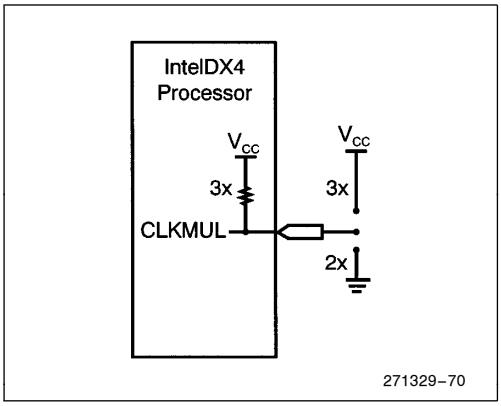


Figure 9-3. Voltage Detect (VOLDET) Sense Pin

To determine which clock multiplier is desired, the IntelDX4 processor samples the status of CLKMUL while RESET is active. If the CLKMUL input is driven low during RESET, the frequency of the core will be twice the external bus frequency (speed doubled mode). If driven high or left floating, speed tripled mode is selected. (See Table 9-1.) In order to allow maximum flexibility, CLKMUL can be jumper-configurable to either V_{CC} (speed tripled mode) or V_{SS} (speed doubled mode). (See Figure 9-3.)

Table 9-1. Clock Multiplier Selection

CLKMUL at RESET	Clock Multiplier	External Clock Freq. (MHz)	Internal Clock Freq. (MHz)
V_{CC} or Not Driven	3	25 33	75 100
V_{SS}	2	25 33	50 66

The clock multiplier selection method is fully backward compatible with Military Intel486 processor-based system designs. The CLKMUL signal occupies a pin which is labeled as an 'INC' on other Military Intel486 processors. Therefore, this pin is not driven in other Military Intel486 processor system designs. The IntelDX4 processor contains an internal pull-up resistor on the CLKMUL signal. As shown in Table 9-1, when CLKMUL is not driven, the internal core frequency defaults to speed tripled mode.

The internal pull-up resistor on the CLKMUL pin is disabled while the IntelDX4 processor is in the Stop Grant or Stop Clock modes. This prevents a low level DC current path from drawing current while in the Stop Grant or Stop Clock states on a system with CLKMUL connected to V_{SS}.

9.2.3 ADDRESS BUS (A31–A2, BE0#–BE3#)

A31–A2 and BE0#–BE3# form the address bus and provide physical memory and I/O port addresses. The Military Intel486 processor is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 Kbytes of I/O address space (00000000H through 0000FFFFH). A31–A2 identify addresses to a 4-byte location. BE0#–BE3# identify which bytes within the 4-byte location are involved in the current transfer.

Addresses are driven back into the Military Intel486 processor over A31–A4 during cache line invalidations. The address lines are active HIGH. When used as inputs into the processor, A31–A4 must meet the setup and hold times, t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.

The byte enable outputs, BE0#–BE3#, determine which bytes must be driven valid for read and write cycles to external memory.

- BE3# applies to D24–D31
- BE2# applies to D16–D23
- BE1# applies to D8–D15
- BE0# applies to D0–D7

BE0#–BE3# can be decoded to generate A0, A1 and BHE# signals used in 8- and 16-bit systems (see Table 10-5). BE0#–BE3# are active LOW and are not driven during bus hold.

9.2.4 DATA LINES (D31–D0)

The bidirectional lines, D31–D0, form the data bus for the Military Intel486 processor D0–D7 define the least significant byte and D24–D31 the most significant byte. Data transfers to 8- or 16-bit devices are possible using the data bus sizing feature controlled by the BS8# or BS16# input pins. D31–D0 are active HIGH. For reads, D31–D0 must meet the setup and hold times, t_{22} and t_{23}. D31–D0 are not driven during read cycles and bus hold.

9.2.5 PARITY

Data Parity Input/Outputs (DP0–DP3)

DP0–DP3 are the data parity pins for the processor. There is one pin for each byte of the data bus. Even parity is generated or checked by the parity generators/checkers. Even parity means that there are an even number of HIGH inputs on the eight corresponding data bus pins and parity pin.

Data parity is generated on all write data cycles with the same timing as the data driven by the Military Intel486 processor. Even parity information must be driven back to the Military Intel486 processor on these pins with the same timing as read information to insure that the correct parity check status is indicated by the Military Intel486 processor.

The values read on these pins do not affect program execution. It is the responsibility of the system to take appropriate actions if a parity error occurs.

Input signals on DP0–DP3 must meet setup and hold times t_{22} and t_{23} for proper operation.

Parity Status Output (PCHK#)

Parity status is driven on the PCHK# pin, and a parity error is indicated by this pin being LOW. PCHK# is driven the clock after ready for read operations to indicate the parity status for the data sampled at the end of the previous clock. Parity is checked during code reads, memory reads and I/O reads. Parity is not checked during interrupt acknowledge cycles. PCHK# only checks the parity status for enabled bytes as indicated by the byte enable and bus size signals. It is valid only in the clock immediately after read data is returned to the Military Intel486 processor. At all other times it is inactive (HIGH). PCHK# is never floated.

Driving PCHK# is the only effect that bad input parity has on the Military Intel486 processor. The Military Intel486 processor will not vector to a bus error interrupt when bad data parity is returned. In systems that will not employ parity, PCHK# can be ignored. In systems not using parity, DP0–DP3 should be connected to V_{CC} through a pull-up resistor.

9.2.6 BUS CYCLE DEFINITION

M/IO#, D/C#, W/R# Outputs

M/IO#, D/C# and W/R# are the primary bus cycle definition signals. They are driven valid as the ADS# signal is asserted. M/IO# distinguishes between memory and I/O cycles, D/C# distinguishes between data and control cycles and W/R# distinguishes between write and read cycles.

Bus cycle definitions as a function of M/IO#, D/C# and W/R# are given in Table 9-2. Note there is a difference between the Military Intel486 processor and Intel386™ processor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the Military Intel486 processor from location 101 in the Intel386 processor. Location 101 is now reserved and will never be generated by the Military Intel486 processor.

Special bus cycles are discussed in section 10.2.11, ‘‘Special Bus Cycles’’.

Table 9-2. ADS# Initiated Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Bus Lock Output (LOCK#)

LOCK# indicates that the Military Intel486 processor is running a read-modify-write cycle where the external bus must not be relinquished between the read and write cycles. Read-modify-write cycles are used to implement memory-based semaphores. Multiple reads or writes can be locked.

When LOCK# is asserted, the current bus cycle is locked and the Military Intel486 processor should be allowed exclusive access to the system bus. LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after ready is returned indicating the last locked bus cycle.

The Military Intel486 processor will not acknowledge bus hold when LOCK# is asserted (though it will allow an address hold). LOCK# is active LOW and is floated during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active. Refer to section 10.2.6, ''Locked Cycles,'' for a detailed discussion of Locked bus cycles.

Pseudo-Lock Output (PLOCK#)

The pseudo-lock feature allows atomic reads and writes of memory operands greater than 32 bits. These operands require more than one cycle to transfer. The Military Intel486 processor asserts PLOCK# during segment table descriptor reads (64 bits) and cache line fills (128 bits).

When PLOCK# is asserted no other master will be given control of the bus between cycles. A bus hold request (HOLD) is not acknowledged during pseudo-locked reads and writes, with one exception. During non-cacheable non-bursted code prefetches, HOLD is recognized on memory cycle boundaries even though PLOCK# is asserted. The Military Intel486 processor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether BRDY# or RDY# are returned.

A pseudo-locked transfer is meaningful only if the memory operand is aligned and if its completely contained within a single cache line.

Because PLOCK# is a function of the bus size and KEN# inputs, PLOCK# should be sampled only in the clock ready is returned. This pin is active LOW and is not driven during bus hold. Refer to section 10.2.7, ''Pseudo-Locked Cycles.''

9.2.6.1 PLOCK# Floating Point Considerations

For processors with an on-chip FPU, the following must be noted for PLOCK# operation. A 64-bit floating point number must be aligned to an 8-byte boundary to guarantee an atomic access. Normally PLOCK# and BLAST# are inverse of each other. However, during the first cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted. Military Intel486 processors with on-chip FPUs also assert PLOCK# during floating point long reads and writes (64 bits), segmentable description reads (64 bits) and code line fills (128 bits).

9.2.7 BUS CONTROL

The bus control signals allow the Military Intel486 processor to indicate when a bus cycle has begun, and allow other system hardware to control burst cycles, data bus width and bus cycle termination.

Address Status Output (ADS#)

The ADS# output indicates that the address and bus cycle definition signals are valid. This signal will go active in the first clock of a bus cycle and go inactive in the second and subsequent clocks of the cycle. ADS# is also inactive when the bus is idle.

ADS# is used by the external bus circuitry as the indication that the Military Intel486 processor has started a bus cycle. The external circuit must sample the bus cycle definition pins on the next rising edge of the clock after ADS# is driven active.

ADS# is active LOW and is not driven during bus hold.

Non-burst Ready Input (RDY#)

RDY# indicates that the current bus cycle is complete. In response to a read, RDY# indicates that the external system has presented valid data on the data pins. In response to a write request, RDY# indicates that the external system has accepted the Military Intel486 processor data. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. Because RDY# is sampled during address hold, data can be returned to the processor when AHOLD is active.

RDY# is active LOW, and is not provided with an internal pull-up resistor. This input must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

9.2.8 BURST CONTROL

Burst Ready Input (BRDY#)

BRDY# performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the Military Intel486 processor data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle.

During a burst cycle, BRDY# will be sampled each clock, and if active, the data presented on the data bus pins will be strobed into the Military Intel486 processor. ADS# is negated during the second through last data cycles in the burst, but address lines A2–A3 and byte enables will change to reflect the next data item expected by the Military Intel486 processor.

If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. An additional complete bus cycle will be initiated after an aborted burst cycle if the cache line fill was not complete. BRDY# is treated as a normal ready for the last data cycle in a burst transfer or for non-burstable cycles. Refer to section 10.2.2, ''Multiple and Burst Cycle Bus Transfers,'' for burst cycle timing.

BRDY# is active LOW and is provided with a small internal pull-up resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.

Burst Last Output (BLAST#)

BLAST# indicates that the next time BRDY# is returned it will be treated as a normal RDY#, terminating the line fill or other multiple-data-cycle transfer. BLAST# is active for all bus cycles regardless of whether they are cacheable or not. This pin is active LOW and is not driven during bus hold.

9.2.9 INTERRUPT SIGNALS

The interrupt signals can interrupt or suspend execution of the processor's current instruction stream.

Reset Input (RESET)

The RESET input must be used at power-up to initialize the processor. The Reset input forces the processor to begin execution at a known state. The processor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper DC and AC specifications. The RESET pin should remain active during this time to ensure proper processor operation. However, for warm boot-ups RESET should remain active for at least 15 CLK periods. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.

RESET will reset SMBASE to the default value of 30000H. If SMBASE relocation is not used, the RESET signal can be used as the only reset. (See section 8, ''System Management Mode Architecture.'')

The Military Intel486 processor will be placed in the Power Down Mode if UP# is sampled active at the falling edge of RESET.

Soft Reset Input (SRESET)

The SRESET (Soft RESET) input, has the same functions as RESET, but does not change the SMBASE, and UP# is not sampled on the falling edge of SRESET. If SMBASE relocation is used by the system, the soft resets should be handled using the SRESET input. The SRESET signal should not be used for the cold boot-up power-on reset.

The SRESET input pin is provided to save the status of SMBASE during Intel 286 processor-compatible mode change. SRESET leaves the location of

SMBASE intact while resetting other units, including the on-chip cache. For compatibility, the system should use SRESET to flush the on-chip cache. The FLUSH# input pin should be used to flush the on-chip cache. SRESET should not be used to initiate test modes.

System Management Interrupt Request Input (SMI#)

SMI# is the system management mode interrupt request signal. The SMI# request is acknowledged by the SMIACT# signal. After the SMI# interrupt is recognized, the SMI# signal will be masked internally until the RSM instruction is executed and the interrupt service routine is complete. SMI# is falling-edge sensitive after internal synchronization.

The SMI# input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. SMI# is provided with a pull-up resistor to maintain compatibility with designs which do not use this feature. SMI# is an asynchronous signal, but setup and hold times, t_{20} and t_{21}, must be met in order to guarantee recognition on a specific clock.

System Management Mode Active Output (SMIACT#)

SMIACT# indicates that the processor is operating in System Management Mode. The processor asserts SMIACT# in response to an SMI interrupt request on the SMI# pin. SMIACT# is driven active after the processor has completed all pending write cycles (including emptying the write buffers), and before the first access to SMRAM when the processor saves (writes) its state (or context) to SMRAM. SMIACT# remains active until the last access to SMRAM when the processor restores (reads) its state from SMRAM. The SMIACT# signal does not float in response to HOLD. The SMIACT# signal is used by the system logic to decode SMRAM.

Maskable Interrupt Request Input (INTR)

INTR indicates that an external interrupt has been generated. Interrupt processing is initiated if the IF flag is active in the EFLAGS register.

The Military Intel486 processor will generate two locked interrupt acknowledge bus cycles in response to asserting the INTR pin. An 8-bit interrupt number will be latched from an external interrupt controller at the end of the second interrupt acknowledge cycle. INTR must remain active until the interrupt acknowledges have been performed to assure program interruption. Refer to section 10.2.10, ''Interrupt Acknowledge,'' for a detailed discussion of interrupt acknowledge cycles.

The INTR pin is active HIGH and is not provided with an internal pull-down resistor. INTR is asynchronous, but the INTR setup and hold times, t_{20} and t_{21}, must be met to assure recognition on any specific clock.

Non-maskable Interrupt Request Input (NMI)

NMI is the non-maskable interrupt request signal. Asserting NMI causes an interrupt with an internally supplied vector value of 2. External interrupt acknowledge cycles are not generated because the NMI interrupt vector is internally generated. When NMI processing begins, the NMI signal will be masked internally until the IRET instruction is executed.

NMI is rising edge sensitive after internal synchronization. NMI must be held LOW for at least four CLK periods before this rising edge for proper operation. NMI is not provided with an internal pull-down resistor. NMI is asynchronous but setup and hold times, t_{20} and t_{21} must be met to assure recognition on any specific clock.

Stop Clock Interrupt Request Input (STPCLK#)

The Military Intel486 processor provides an interrupt mechanism, STPCLK#, that allows system hardware to control the power consumption of the processor by stopping the internal clock (output of the PLL) to the processor core in a controlled manner. This low-power state is called the Stop Grant state. In addition, the STPCLK# interrupt allows the system to change the input frequency within the specified range or completely stop the CLK input frequency (input to the PLL). If the CLK input is completely stopped, the processor enters into the Stop Clock state–the lowest power state. If the frequency is changed or stopped, the Military Intel486 processor will not return to the Stop Grant state until the CLK input has been running at a constant frequency for the time period necessary to stabilize the PLL (minimum of 1 ms).

The Military Intel486 processor will generate a Stop Grant bus cycle in response to the STPCLK#

interrupt request. STPCLK# is active LOW and is provided with an internal pull-up resistor. STPCLK# is an asynchronous signal, but must remain active until the processor issues the Stop Grant bus cycle. (Refer to section 10.2.11.3, ''Stop Grant Indication Cycle.'')

9.2.10 BUS ARBITRATION SIGNALS

This section describes the mechanism by which the processor relinquishes control of its local bus when requested by another bus master.

Bus Request Output (BREQ)

The Military Intel486 processor asserts BREQ whenever a bus cycle is pending internally. Thus, BREQ is always asserted in the first clock of a bus cycle, along with ADS#. Furthermore, if the Military Intel486 processor is currently not driving the bus (due to HOLD, AHOLD, or BOFF#), BREQ is asserted in the same clock that ADS# would have been asserted if the Military Intel486 processor were driving the bus. After the first clock of the bus cycle, BREQ may change state. It will be asserted if additional cycles are necessary to complete a transfer (via BS8#, BS16#, KEN#), or if more cycles are pending internally. However, if no additional cycles are necessary to complete the current transfer, BREQ can be negated before ready comes back for the current cycle. External logic can use the BREQ signal to arbitrate among multiple processors. This pin is driven regardless of the state of bus hold or address hold. BREQ is active HIGH and is never floated. During a hold state, internal events may cause BREQ to be de-asserted prior to any bus cycles.

Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the Military Intel486 processor bus. The Military Intel486 processor will respond to an active HOLD signal by asserting HLDA and placing most of its output and input/output pins in a high impedance state (floated) after completing its current bus cycle, burst cycle, or sequence of locked cycles. In addition, if the Military Intel486 processor receives a HOLD request while performing a code fetch, and that cycle is backed off (BOFF#), the Military Intel486 processor will recognize HOLD before restarting the cycle. The code fetch can be non-cacheable or cacheable and non-bursted or bursted. The BREQ, HLDA, PCHK# and FERR# pins are not floated during bus hold. The Military Intel486 processor will maintain its bus in this state until the HOLD is de-asserted. Refer to section 10.2.9, ''Bus Hold,'' for timing diagrams for bus hold cycles and HOLD request acknowledge during BOFF#.

Unlike the Intel386 processor, the Military Intel486 processor will recognize HOLD during reset. Pull-up resistors are not provided for the outputs that are floated in response to HOLD. HOLD is active HIGH and is not provided with an internal pull-down resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper chip operation.

Bus Hold Acknowledge Output (HLDA)

HLDA indicates that the Military Intel486 processor has given the bus to another local bus master. HLDA goes active in response to a hold request presented on the HOLD pin. HLDA is driven active in the same clock that the Military Intel486 processor floats its bus.

HLDA will be driven inactive when leaving bus hold and the Military Intel486 processor will resume driving the bus. The Military Intel486 processor will not cease internal activity during bus hold because the internal cache will satisfy the majority of bus requests. HLDA is active HIGH and remains driven during bus hold.

Backoff Input (BOFF#)

Asserting the BOFF# input forces the Military Intel486 processor to release control of its bus in the next clock. The pins floated are exactly the same as in response to HOLD. The response to BOFF# differs from the response to HOLD in two ways: First, the bus is floated immediately in response to BOFF# while the Military Intel486 processor completes the current bus cycle before floating its bus in response to HOLD. Second the Military Intel486 processor does not assert HLDA in response to BOFF#.

The Military Intel486 processor remains in bus hold until BOFF# is negated. Upon negation, the Military Intel486 processor restarts the bus cycle aborted when BOFF# was asserted. To the internal execution engine the effect of BOFF# is the same as inserting a few wait states to the original cycle. Refer to section 10.2.12, ''Bus Cycle Restart,'' for a description of bus cycle restart.

Any data returned to the Military Intel486 processor while BOFF# is asserted is ignored. BOFF# has higher priority than RDY# or BRDY#. If both BOFF# and ready are returned in the same clock, BOFF# takes effect. If BOFF# is asserted while the bus is idle, the Military Intel486 processor will float its bus in the next clock. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper chip operation.

9.2.11 CACHE INVALIDATION

The AHOLD and EADS# inputs are used during cache invalidation cycles. AHOLD conditions the Military Intel486 processor address lines, A4–A31, to accept an address input. EADS# indicates that an external address is actually valid on the address inputs. Activating EADS# will cause the Military Intel486 processor to read the external address bus and perform an internal cache invalidation cycle to the address indicated. Refer to section 10.2.8, ''Invalidate Cycle,'' or cache invalidation cycle timing.

Address Hold Request Input (AHOLD)

AHOLD is the address hold request. It allows another bus master access to the Military Intel486 processor address bus for performing an internal cache invalidation cycle. Asserting AHOLD will force the Military Intel486 processor to stop driving its address bus in the next clock. While AHOLD is active only the address bus will be floated, the remainder of the bus can remain active. For example, data can be returned for a previously specified bus cycle when AHOLD is active. The Military Intel486 processor will not initiate another bus cycle during address hold. Because the Military Intel486 processor floats its bus immediately in response to AHOLD, an address hold acknowledge is not required. If AHOLD is asserted while a bus cycle is in progress, and no readies are returned during the time AHOLD is asserted, the Military Intel486 processor will redrive the same address (that it originally sent out) once AHOLD is negated.

AHOLD is recognized during reset. Because the entire cache is invalidated by reset, any invalidation cycles run during reset will be unnecessary. AHOLD is active HIGH and is provided with a small internal pull-down resistor. It must satisfy the setup and hold times t_{18} and t_{19} for proper chip operation. AHOLD also determines whether or not the built in self test features of the Military Intel486 processor will be exercised on assertion of RESET. (See section 11.1, ''Built-In Self Test.'')

External Address Valid Input (EADS#)

EADS# indicates that a valid external address has been driven onto the Military Intel486 processor address pins. This address will be used to perform an internal cache invalidation cycle. The external address will be checked with the current cache contents. If the address specified matches any areas in the cache, that area will immediately be invalidated.

An invalidation cycle may be run by asserting EADS# regardless of the state of AHOLD, HOLD and BOFF#. EADS# is active LOW and is provided with an internal pull-up resistor. EADS# must satisfy the setup and hold times t_{12} and t_{13} for proper chip operation.

9.2.12 CACHE CONTROL

Cache Enable Input (KEN#)

KEN# is the cache enable pin. KEN# is used to determine whether the data being returned by the current cycle is cacheable. When KEN# is active and the Military Intel486 processor generates a cycle that can be cached (most any memory read cycle), the cycle will be transformed into a cache line fill cycle.

A cache line is 16 bytes long. During the first cycle of a cache line fill the byte-enable pins should be ignored and data should be returned as if all four byte enables were asserted. The Military Intel486 processor will run between 4 and 16 contiguous bus cycles to fill the line depending on the bus data width selected by BS8# and BS16#. Refer to section 10.2.3, ''Cacheable Cycles,'' for a description of cache line fill cycles.

The KEN# input is active LOW and is provided with a small internal pull-up resistor. It must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

Cache Flush Input (FLUSH#)

The FLUSH# input forces the Military Intel486 processor to flush its entire internal cache. FLUSH# is active LOW and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition on any specific clock.

FLUSH# also determines whether or not the tri-state test mode of the Military Intel486 processor will be invoked on assertion of RESET. (See section 11.4, ''Tri-State Output Test Mode.'')

9.2.13 PAGE CACHEABILITY (PWT, PCD)

The PWT and PCD output signals correspond to two user attribute bits in the page table entry. When paging is enabled, PWT and PCD correspond to bits 3 and 4 of the page table entry respectively. For cycles that are not paged when paging is enabled (for example I/O cycles) PWT and PCD correspond to bits 3 and 4 in control register 3. When paging is disabled, the Military Intel486 processor ignores the PCD and PWT bits and assumes they are zero for the purpose of caching and driving PCD and PWT.

PCD is masked by the CD (cache disable) bit in control register 0 (CR0). When CD = 1 (cache line fills disabled) the Military Intel486 processor forces PCD HIGH. When CD = 0, PCD is driven with the value of the page table entry/directory.

The purpose of PCD is to provide a cacheable/non-cacheable indication on a page by page basis. The Military Intel486 processor will not perform a cache fill to any page in which bit 4 of the page table entry is set. PWT corresponds to the write-back bit and can be used by an external cache to provide this functionality. PCD and PWT bits are assigned to be zero during real mode or whenever paging is disabled. Refer to section 7.6, ''Page Cacheability,'' for a discussion of non-cacheable pages.

PCD and PWT have the same timing as the cycle definition pins (M/IO#, D/C#, W/R#). PCD and PWT are active HIGH and are not driven during bus hold.

9.2.14 NUMERIC ERROR REPORTING (FERR#, IGNNE#)

To allow PC-type floating point error reporting, Military Intel486 DX, IntelDX2, and IntelDX4 processors provide two pins, FERR# and IGNNE#.

Floating Point Error Output (FERR#)

The processor asserts FERR# whenever an unmasked floating point error is encountered. FERR# is similar to the ERROR# pin on the Intel387 math coprocessor. FERR# can be used by external logic for PC-type floating point error reporting in systems with a Military Intel486 DX, IntelDX2, or IntelDX4 processor. FERR# is active LOW and is not floated during bus hold.

In some cases, FERR# is asserted when the next floating point instruction is encountered. In other cases, it is asserted before the next floating point instruction is encountered, depending on the execution state of the instruction that caused the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction):

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction:

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

Ignore Numeric Error Input (IGNNE#)

Military Intel486 DX, IntelDX2, and IntelDX4 processors will ignore a numeric error and continue executing non-control floating point instructions when IGNNE# is asserted, and FERR# is still activated. When de-asserted, the processor will freeze on a non-control floating point instruction if a previous instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set.

The IGNNE# input is active LOW and provided with a small internal pull-up resistor. This input is asynchronous, but must meet setup and hold times t_{20} and t_{21} to insure recognition on any specific clock.

9.2.15 BUS SIZE CONTROL (BS16#, BS8#)

The BS16# and BS8# inputs allow external 16- and 8-bit buses to be supported with a small number of external components. The Military Intel486 processor samples these pins every clock. The value sampled in the clock before ready determines the bus size. When asserting BS16# or BS8# only 16 or 8 bits of the data bus need be valid. If both BS16# and BS8# are asserted, an 8-bit bus width is selected.

When BS16# or BS8# are asserted, the Military Intel486 processor will convert a larger data request to the appropriate number of smaller transfers. The byte enables will also be modified appropriately for the bus size selected.

BS16# and BS8# are active LOW and are provided with small internal pull-up resistors. BS16# and BS8# must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

9.2.16 ADDRESS BIT 20 MASK (A20M#)

Asserting the A20M# input causes the Military Intel486 processor to mask physical address bit 20 before performing a lookup in the internal cache and before driving a memory cycle to the outside world. When A20M# is asserted, the Military Intel486 processor emulates the 1-Mbyte address wraparound that occurs on the 8086. A20M# is active LOW and must be asserted only when the processor is in real mode. The A20M# is not defined in Protected Mode. A20M# is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For correct operation of the chip, A20M# should not be active at the falling edge of RESET.

A20M# exhibits a minimum 4 clock latency, from time of assertion to masking of the A20 bit. A20M# is ignored during cache invalidation cycles. I/O writes require A20M# to be asserted a minimum of 2 clocks prior to RDY being returned for the I/O write. This insures recognition of the address mask before the Military Intel486 processor begins execution of the instruction following OUT. If A20M# is asserted after the ADS# of a data cycle, the A20 address signal is not masked during this cycle but is masked in the next cycle. During a prefetch (cacheable or not), if A20M# is asserted after the first ADS#, A20 is not masked for the duration of the prefetch; even if BS16# or BS8# is asserted.

9.2.17 INTELDX4 PROCESSOR VOLTAGE DETECT SENSE OUTPUT (VOLDET)

A voltage detect sense pin (VOLDET) has been added to the IntelDX4 processor PGA package. This pin allows external system logic to distinguish between a 5V Military Intel486 DX or IntelDX2 processor and the 3.3V IntelDX4 processor. The pin passively indicates to external logic whether the installed PGA processor requires 5V (in the case of the Military Intel486 DX or IntelDX2 processor) or 3.3V (in the case of the IntelDX4 processor). Pin S4 has been defined as the VOLDET pin because this pin is defined as an NC pin on the Military Intel486 DX and IntelDX2 processor.

To utilize this feature, a weak, external pull-up resistor should be connected to the VOLDET pin. This pin samples high (logic 1) if the installed processor is a 5V Military Intel486 DX or IntelDX2 processor. This pin samples low (logic 0) if a IntelDX4 processor is installed. Upon sampling the logic level of this pin, external logic can then enable the proper V_{CC} level to the processor. In power sensitive applications, an active element is preferred for the pull-up device because it could be disabled after sampling, thereby eliminating the resulting DC current path when the installed processor is the IntelDX4 processor.

Figure 9-4 shows a logical representation of the Voltage Detect sense mechanism.

This pin can remain not connected for those system designs that do not wish to utilize this voltage detect feature.

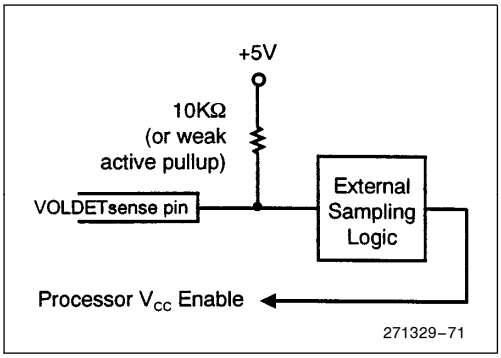


Figure 9-4. Voltage Detect (VOLDET) Sense Pin

9.2.18 BOUNDARY SCAN TEST SIGNALS

Test Clock (TCK)

TCK is an input to the Military Intel486 processor and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.

In addition to using TCK as a free running clock, it may be stopped in a low, O, state, indefinitely as described in IEEE 1149.1. While TCK is stopped in the low state, the boundary scan latches retain their state.

When boundary scan is not used, TCK should be tied high or left as a NC. (This is important during power up to avoid the possibility of glitches on the TCK which could prematurely initiate boundary scan operations.) TCK is supplied with an internal pull-up resistor.

TCK is a clock signal and is used as a reference for sampling other JTAG signals. On the rising edge of TCK, TMS and TDI are sampled. On the falling edge of TCK, TDO is driven.

Test Mode Select (TMS)

TMS is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic, as described in section 11.5.4, ‘‘Test Access Port Controller.’’

To guarantee deterministic behavior of the TAP controller TMS is provided with an internal pull-up resistor. If boundary scan is not used, TMS may be tied high or left unconnected. TMS is sampled on the rising edge of TCK. TMS is used to select the internal TAP states required to load boundary scan instructions to data on TDI. For proper initialization of the JTAG logic, TMS should be driven high, ‘‘1,’’ for at least four TCK cycles following the rising edge of RESET.

Test Data Input (TDI)

TDI is the serial input used to shift JTAG instructions and data into the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR TAP controller states, respectively. These states are selected using the TMS signal as described in section 11.5.4, ‘‘Test Access Port Controller.’’

An internal pull-up resistor is provided on TDI to ensure a known logic state if an open circuit occurs on the TDI path. Note that when ‘‘1’’ is continuously shifted into the instruction register, the BYPASS instruction is selected. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR states. During all other TAP controller states, TDI is a ‘‘don’t care.’’ TDI is only sampled when TMS and TCK have been used to select the SHIFT-IR or SHIFT-DR states in the TAP controller. For proper initialization of JTAG logic, TDI should be driven high, ‘‘1,’’ for at least four TCK cycles following the rising edge of RESET.

Test Data Output (TDO)

TDO is the serial output used to shift JTAG instructions and data out of the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR TAP controller states, respectively. These states are selected using the TMS signal as described in section 11.5.4, ‘‘Test Access Port Controller’’. When not in SHIFT-IR or SHIFT-DR state, TDO is driven to a high impedance state to allow connecting TDO of different devices in parallel. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state. TDO is only driven when TMS and TCK have been used to select the SHIFT-IR or SHIFT-DR states in the TAP controller.

9.3 Interrupt and Non-Maskable Interrupt Interface

The Military Intel486 processor provides four asynchronous interrupt inputs: INTR (interrupt request), NMI (non-maskable interrupt input), SMI# (system management interrupt) and STPCLK# (stop clock interrupt). This section describes the hardware interface between the instruction execution unit and the pins. For a description of the algorithmic response to interrupts refer to section 4.8, ''Interrupts''. For interrupt timings refer to section 10.2.10, ''Interrupt Acknowledge''.

9.3.1 INTERRUPT LOGIC

The Military Intel486 processor contains a two-clock synchronizer on the interrupt line. An interrupt request will reach the internal instruction execution unit two clocks after the INTR pin is asserted, if proper setup is provided to the first stage of the synchronizer.

There is no special logic in the interrupt path other than the synchronizer. The INTR signal is level sensitive and must remain active for the instruction execution unit to recognize it. The interrupt will not be serviced by the Military Intel486 processor if the INTR signal does not remain active.

The instruction execution unit will look at the state of the synchronized interrupt signal at specific clocks during the execution of instructions (if interrupts are enabled). These specific clocks are at instruction boundaries, or iteration boundaries in the case of string move instructions. Interrupts will only be accepted at these boundaries.

An interrupt must be presented to the Military Intel486 processor INTR pin three clocks before the end of an instruction for the interrupt to be acknowledged. Presenting the interrupt 3 clocks before the end of an instruction allows the interrupt to pass through the two clock synchronizer leaving one clock to prevent the initiation of the next sequential instruction and to begin interrupt service. If the interrupt is not received in time to prevent the next instruction, it will be accepted at the end of next instruction, assuming INTR is still held active.

The longest latency between when an interrupt request is presented on the INTR pin and when the interrupt service begins is: longest instruction used + the two clocks for synchronization + one clock required to vector into the interrupt service microcode.

9.3.2 NMI LOGIC

The NMI pin has a synchronizer like that used on the INTR line. Other than the synchronizer, the NMI logic is different from that of the maskable interrupt.

NMI is edge triggered as opposed to the level triggered INTR signal. The rising edge of the NMI signal is used to generate the interrupt request. The NMI input need not remain active until the interrupt is actually serviced. The NMI pin only needs to remain active for a single clock if the required setup and hold times are met. NMI will operate properly if it is held active for an arbitrary number of clocks.

The NMI input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent NMI may not be generated if the NMI is not held inactive for at least four clocks after being asserted.

The NMI input is internally masked whenever the NMI routine is entered. The NMI input will remain masked until an IRET (return from interrupt) instruction is executed. Masking the NMI signal prevents recursive NMI calls. If another NMI occurs while the NMI is masked off, the pending NMI will be executed after the current NMI is done. Only one NMI can be pending while NMI is masked.

9.3.3 SMI# LOGIC

SMI# is edge triggered like NMI, but the interrupt request is generated on the falling-edge. SMI# is an asynchronous signal, but setup and hold times, t_{20} and t_{21}, must be met in order to guarantee recognition on a specific clock. The SMI# input need not remain active until the interrupt is actually serviced. The SMI# input only needs to remain active for a single clock if the required setup and hold times are met. SMI# will also work correctly if it is held active for an arbitrary number of clocks.

The SMI# input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent SMI# might not be recognized if the SMI# input is not held inactive for at least four clocks after being asserted.

SMI#, like NMI, is not affected by the IF bit in the EFLAGS register and is recognized on an instruction boundary. An SMI# will not break locked bus cycles.

The SMI# has a higher priority than NMI and is not masked during an NMI.

After the SMI# interrupt is recognized, the SMI# signal will be masked internally until the RSM instruction is executed and the interrupt service routine is complete. Masking the SMI# prevents recursive SMI# calls. The SMI# input must be de-asserted for at least 4 clocks to reset the edge triggered logic. If another SMI# occurs while the SMI# is masked, the pending SMI# will be recognized and executed on the next instruction boundary after the current SMI# completes. This instruction boundary occurs before execution of the next instruction in the interrupted application code, resulting in back to back SMM handlers. Only one SMI# can be pending while SMI# is masked.

The SMI# signal is synchronized internally and should be asserted at least three (3) CLK periods prior to asserting the RDY# signal in order to guarantee recognition on a specific instruction boundary. This is important for servicing an I/O trap with an SMI# handler.

9.3.4 STPCLK# LOGIC

STPCLK# is level triggered and active LOW. STPCLK# is an asynchronous signal, but must remain active until the processor issues the Stop Grant bus cycle. STPCLK# may be de-asserted at any time after the processor has issued the Stop Grant bus cycle. When the processor enters the Stop Grant state, the internal pull-up resistor of STPCLK#, CLKMUL (for IntelDX4 processor), and UP# are disabled so that the processor power consumption is reduced. The STPCLK# input must be driven high (not floated) in order to exit the Stop Grant state. **STPCLK# must be de-asserted for a minimum of 5 clocks after RDY# or BRDY# is returned active for the Stop Grant Bus Cycle before being asserted again.**

When the processor recognizes a STPCLK# interrupt, the processor will stop execution on the next instruction boundary (unless superseded by a higher priority interrupt), stop the pre-fetch unit, empty all internal pipelines and the write buffers, generate a Stop Grant bus cycle, and then stop the internal clock. At this point the processor is in the Stop Grant state.

The processor cannot respond to a STPCLK# request from an HLDA state because it cannot empty the write buffers and, therefore, cannot generate a Stop Grant cycle.

The rising edge of STPCLK# will tell the processor that it can return to program execution at the instruction following the interrupted instruction.

Unlike the normal interrupts, INTR and NMI, the STPCLK# interrupt does not initiate acknowledge cycles or interrupt table reads. Among external interrupts, the STPCLK# order of priority is shown in section 4.8.

9.4 Write Buffers

The Military Intel486 processor contains four write buffers to enhance the performance of consecutive writes to memory. The buffers can be filled at a rate of one write per clock until all four buffers are filled.

When all four buffers are empty and the bus is idle, a write request will propagate directly to the external bus bypassing the write buffers. If the bus is not available at the time the write is generated internally, the write will be placed in the write buffers and propagate to the bus as soon as the bus becomes available. The write is stored in the on-chip cache immediately if the write is a cache hit.

Writes will be driven onto the external bus in the same order in which they are received by the write buffers. Under certain conditions a memory read will go onto the external bus before the memory writes pending in the buffer even though the writes occurred earlier in the program execution.

A memory read will only be reordered in front of all writes in the buffers under the following conditions: If all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions the Military Intel486 processor will not read from an external memory location that needs to be updated by one of the pending writes.

Reordering of a read with the writes pending in the buffers can only occur once before all the buffers are emptied. Reordering read once only maintains cache consistency. Consider the following example: The processor writes to location X. Location X is in the internal cache, so it is updated there immediately. However, the bus is busy so the write out to main memory is buffered (see Figure 9-5). At this point, any reads to location X would be cache hits and most up-to-date data would be read.

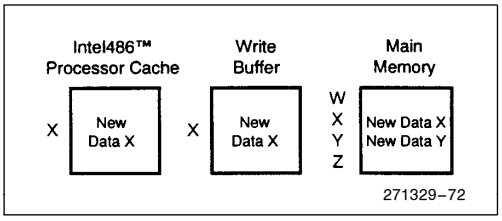


Figure 9-5. Reordering of a Reads with Write Buffers

The next instruction causes a read to location Y. Location Y is not in the cache (a cache miss). Because the write in the write buffer is a cache hit, the read is reordered. When location Y is read, it is put into the cache. The possibility exists that location Y will replace location X in the cache. If this is true, location X would no longer be cached (see Figure 9-6).

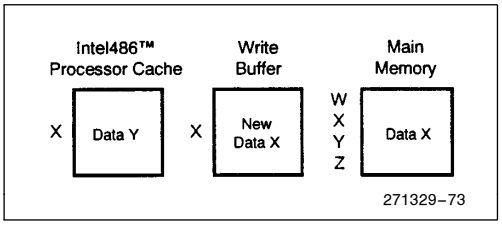


Figure 9-6. Reordering of a Reads with Write Buffers

Cache consistency has been maintained up to this point. If a subsequent read is to location X (now a cache miss) and it was reordered in front of the buffered write to location X, stale data would be read. This is why only 1 read is allowed to be reordered. Once a read is reordered, all the writes in the write buffer are flagged as cache misses to ensure that no more reads are reordered. Because one of the conditions to reorder a read is that all writes in the write buffer must be cache hits, no more reordering is allowed until all of those flagged writes propagate to the bus. Similarly, if an invalidation cycle is run all entries in the write buffer are flagged as cache misses.

For multiple processor systems and/or systems using DMA techniques, such as bus snooping, locked semaphores should be used to maintain cache consistency.

9.4.1 WRITE BUFFERS AND I/O CYCLES

Input/Output (I/O) cycles must be handled in a different manner by the write buffers.

I/O reads are never reordered in front of buffered memory writes. This insures that the Military Intel486 processor will update all memory locations before reading status from an I/O device.

The Military Intel486 processor never buffers single I/O writes. When processing an OUT instruction, internal execution stops until the I/O write actually completes on the external bus. This allows time for the external system to drive an invalidate into the Military Intel486 processor or to mask interrupts before the processor progresses to the instruction following OUT. REP OUTS instructions will be buffered.

A read cycle must be explicitly generated to a non-cacheable location in memory to guarantee that a read bus cycle is performed. This read will not be allowed to proceed to the bus until after the I/O write has completed because I/O writes are not buffered. The I/O device will have time to recover to accept another write during the read cycle.

9.4.2 WRITE BUFFERS IMPLICATIONS ON LOCKED BUS CYCLES

Locked bus cycles are used for read-modify-write accesses to memory. During a read-modify-write access, a memory base variable is read, modified and then written back to the same memory location. It is important that no other bus cycles, generated by other bus masters or by the Military Intel486 processor itself, be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle, the Military Intel486 processor will always access external memory, it will never look for the location in the on-chip cache, but for write cycles, data is written in the internal cache (if cache hit) and in the external memory. All data pending in the Military Intel486 processor's write buffers will be written to memory before a locked cycle is allowed to proceed to the external bus.

The Military Intel486 processor will assert the LOCK# pin after the write buffers are emptied during a locked bus cycle. With the LOCK# pin asserted, the processor will read the data, operate on the data and place the results in a write buffer. The contents of the write buffer will then be written to external memory. LOCK# will become inactive after the write part of the locked cycle.

9.5 Reset and Initialization

The Military Intel486 processor has a built in self test (BIST) that can be run during reset. BIST is invoked if the AHOLD pin is asserted for 1 clock before and 1 clock after RESET is de-asserted. RESET must be active for 15 clocks with or without BIST being enabled. To ensure proper results, neither FLUSH# nor SRESET can be asserted while BIST is executing. Refer to section 11.0, ''Testability,'' for information on Military Intel486 processor testability.

The Military Intel486 processor registers have the values shown in Table 9-3 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) will contain 04 and the lower byte (DL) will contain the revision identifier. (See Table 9-4.)

RESET forces the Military Intel486 processor to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active.

All entries in the cache are invalidated by RESET.

9.5.1 FLOATING POINT REGISTER VALUES

In addition to the register values listed above, Military Intel486 DX, IntelDX2, and IntelDX4 processors have the floating point register values shown in Table 9-5.

The floating point registers are initialized as if the FINIT/FNINIT (initialize processor) instruction was executed if the BIST was performed. If the BIST is not executed, the floating point registers are unchanged.

The Military Intel486 processor will start executing instructions at location FFFFFFF0H after RESET. When the first Inter Segment Jump or Call is executed, address lines A20–A31 will drop LOW for CS-relative memory cycles, and the Military Intel486 processor will only execute instructions in the lower one Mbyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of RESETs.

Table 9-3. Register Values after Reset

Register	Initial Value (BIST)	Initial Value (No BIST)
EAX	Zero (Pass)	Undefined
ECX	Undefined	Undefined
EDX	0400 + Revision ID	0400 + Revision ID
EBX	Undefined	Undefined
ESP	Undefined	Undefined
EBP	Undefined	Undefined
ESI	Undefined	Undefined
EDI	Undefined	Undefined
EFLAGS	00000002h	00000002h
EIP	0FFF0h	0FFF0h
ES	0000h	0000h
CS	F000h*	F000h*
SS	0000h	0000h
DS	0000h	0000h
FS	0000h	0000h
GS	0000h	0000h
IDTR	Base = 0, Limit = 3FFh	Base = 0, Limit = 3FFh
CR0	60000010h	60000010h
DR7	00000000h	00000000h

Table 9-4. Military Intel486™ Processor Revision ID

Product	Component ID (DH)	Revision ID (DL)
Military Intel486™ DX Processor	04	0x 1x
IntelDX2™ Processor	04	3x
IntelDX4™ Processor	04	8x

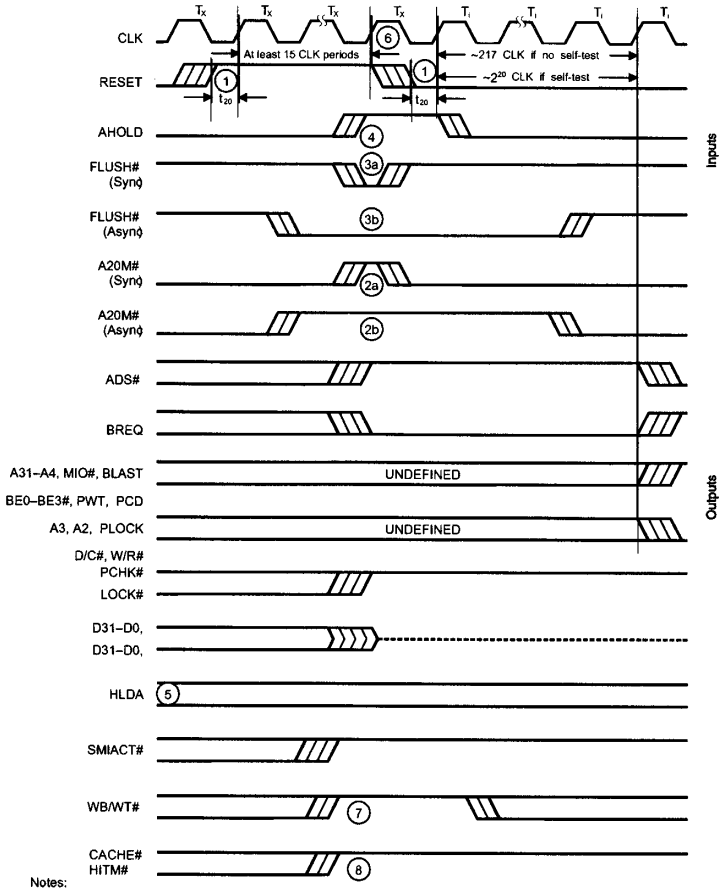
Table 9-5. Floating Point Values after Reset

Register	Initial Value (BIST)	Initial Value (No BIST)
CW	037Fh	Unchanged
SW	0000h	Unchanged
TW	FFFFh	Unchanged
FIP	00000000h	Unchanged
FEA	00000000h	Unchanged
FCS	0000h	Unchanged
FDS	0000h	Unchanged
FOP	000h	Unchanged
FSTACK	Undefined	Unchanged

9.5.2 PIN STATE DURING RESET

The Military Intel486 processor recognizes and can respond to HOLD, AHOLD, and BOFF# requests regardless of the state of RESET. Thus, even though the processor is in reset, it can still float its bus in response to any of these requests.

While in reset, the Military Intel486 processor bus is in the state shown in Figure 9-7 if the HOLD, AHOLD and BOFF# requests are inactive. The figure shows the bus state for the Military Intel486 processor. Note that the address (A31–A2, BE3#–BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All undefined pins **(except FERR#)** assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address FFFFFFF0H.



Notes:

1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.

2a. When A20M# is driven synchronously, it must be driven high (inactive) for the CLK edge prior to the falling edge of RESET to ensure proper operation. A20M# setup and hold times must be met.

2b. When A20M# is driven asynchronously, it should be driven low (active) for two CLKs prior to and two CLKs after the falling edge of RESET to ensure proper operation.

3a. When FLUSH# is driven synchronously, it must be driven low (high) for the CLK edge prior to the falling edge of RESET to invoke the 3-state Output Test Mode. All outputs are guaranteed 3-stated within 10 CLKs of RESETR being deasserted. FLUSH# setup and hold times must be met.

3b. When FLUSH# is driven asynchronously, it must be driven low (active) for two CLKs period prior to and two CLKs after the falling edge of RESET to invoke the 3-state Output Test Mode. All outputs are guaranteed 3-stated within 10 CLKs of RESET being deasserted.

4. AHOLD should be driven high (active) for the CLK edge prior to the falling edge of RESET to invoke the Built-In-Self-Test (BIST). AHOLD setup and hold times must be met.

5. Hold is recognized normally during RESET. On power-up HLDA is indeterminate until RESET is recognized by the processor.

6. 15 CLKs RESET pulse width for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

7. WB/WT# should be driven high for at least one CLK before falling edge of RESET and at least one CLK after falling edge of RESET to enable the Enhanced Bus Mode. The Standard Bus Mode will be enabled if WB/WT# is sampled low or left floating at the falling edge of RESET.

8. The system may sample HITM# to detect the presence of the Enhanced Bus Mode. If HITM# is HIGH for one CLK after reset is inactive, the Enhanced Bus Mode is present.

271329–74

Figure 9-7. Pin States during RESET

139

9.5.2.1 Controlling the CLK Signal in the Processor during Power On

The power on requirements of the Military Intel486 processor with regards to allowable CLK input during the power on sequence have never been specified. Clocking the processor before Vcc has reached its normal operating level can cause unpredictable results on Military Intel486 processors. While Intel will maintain original clock and power specifications (none), this section reflects what Intel considers to be a good clock design.

Intel strongly recommends that system designers ensure that a clock signal is not presented to the Military Intel486 processor until Vcc has stabilized at its normal operating level. This design recommendation can easily be met by gating the clock signal with a POWERGOOD signal. The POWERGOOD signal should reflect the status of Vcc at the Military Intel486 processor (which may be different from the power supply status in designs that provide power to the processor through the use of a voltage regulator or converter).

Most clock synthesizers and some clock oscillators contain on-board gating logic. If external gating logic is implemented, it should be done on the original clock signal output from the clock oscillator/synthesizer. Gating the clock to the processor independently of the clock to the rest of the motherboard will cause clock skew, which may violate processor or chipset timing requirements. If the clock signal to the motherboard is enabled with a POWERGOOD signal, it is also important to verify that the motherboard logic does not require a clock input prior to this POWERGOOD signal. Some chipsets also gate the clock to the processor only after a POWERGOOD signal, which inherently meets the requirements of this design note. Designs should implement this design note, so as to maintain maximum flexibility with all Military Intel486 processor steppings.

9.5.2.2 FERR# Pin State During Reset for Military Intel486 DX, IntelDX2, and IntelDX4 Processors

FERR# reflects the state of the ES (error summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized. The floating point unit's status word register can be initialized by BIST or by executing FINIT/FNINIT instruction. Thus, after reset and before executing the first FINIT or FNINIT instruction, the values of the FERR# and the numeric status word register bits 0–7 depends on whether or not BIST is performed. Table 9-6 shows the state of FERR# signal after reset and before the execution of the FINIT/FNINIT instruction.

Table 9-6. FERR# Pin State after Reset and before FP Instructions

BIST Performed	FERR# Pin	FPU Status Word Register Bits 0–7
YES	Inactive (High)	Inactive (Low)
NO	Undefined (Low or High)	Undefined (Low or High)

After the first FINIT or FNINIT instruction, FERR# pin and the FPU status word register bits (07) will be inactive irrespective of the Built-In Self-Test (BIST).

9.6 Clock Control

The Military Intel486 processor provides an interrupt mechanism (STPCLK#) that allows system hardware to control the power consumption of the processor by stopping the internal clock (output of the PLL) to the processor core in a controlled manner. This low-power state is called the Stop Grant state. In addition, the STPCLK# interrupt allows the system to change the input frequency within the specified range or completely stop the CLK input frequency (input to the PLL). If the CLK input is completely stopped, the processor enters into the Stop Clock statethe lowest power state.

There are two targets for the low-power mode supply current:

- ~20–100 mA in the **Stop Grant state** (fast wake-up, frequency-and voltage-dependent), and
- ~100–1000 μA in the full **Stop Clock state** (slow wake-up, voltage-dependent).

See section 9.6.3.2 and 9.6.3.3, for a detailed description of the Stop Grant and Stop Clock states.

9.6.1 STOP GRANT BUS CYCLE

A special Stop Grant bus cycle will be driven to the bus after the processor recognizes the STPCLK# interrupt. The definition of this bus cycle is the same as the HALT cycle definition for the standard Military Intel486 processor, with the exception that the Stop Grant bus cycle drives the value 0000 0010H on the address pins. The system hardware must acknowledge this cycle by returning RDY# or BRDY#. **The processor will not enter the Stop Grant state until either RDY# or BRDY# has been returned.**

The Stop Grant bus cycle is defined as follows:

M/IO# = 0, D/C# = 0, W/R# = 1, Address Bus = 0000 0010H (A_4 = 1), BE3#–BE0# = 1011, Data bus = undefined

The latency between a STPCLK# request and the Stop Grant bus cycle is dependent on the current instruction, the amount of data in the processor write buffers, and the system memory performance. (See Figure 9-8.)

9.6.2 PIN STATE DURING STOP GRANT

During the Stop Grant state, most output and input/output signals of the processor will maintain their previous condition (the level they held when entering the Stop Grant state). The data and data parity signals will be tri-stated. In response to HOLD being driven active during the Stop Grant state (when the CLK input is running), the processor will generate HLDA and tri-state all output and input/output signals that are tri-stated during the HOLD/HLDA state. After HOLD is de-asserted all signals will return to their prior state before the HOLD/HLDA sequence.

In order to achieve the lowest possible power consumption during the Stop Grant state, the system designer must ensure the input signals with pull-up resistors are not driven LOW and the input signals with pull-down resistors are not driven HIGH. (See Table 3-11 in the Quick Pin Reference section for signals with internal pull-up and pull-down resistors.)

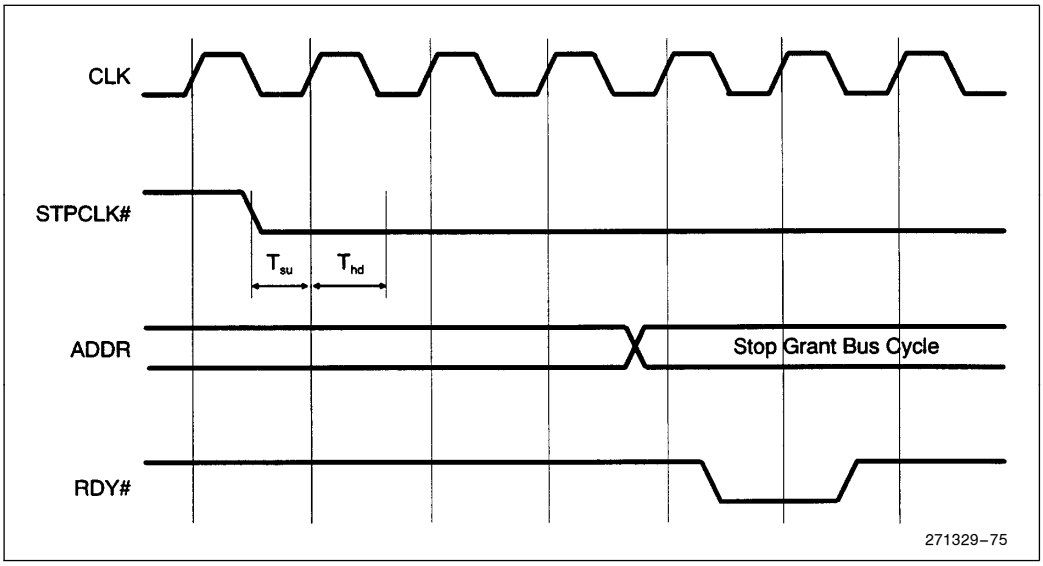


Figure 9-8. Stop Clock Protocol

All inputs, except the data bus pins must be driven to the power supply rails to ensure the lowest possible current consumption during Stop Grant or Stop Clock modes. For compatibility with future processors, data pins should be driven low to achieve the lowest possible power consumption. Pull-down resistors/bus keepers are needed to minimize leakage current.

If HOLD is asserted during the Stop Grant state, all pins that are normally floated during HLDA will still be floated by the processor. The floated pins should be driven to a low level. (See Table 9-7.)

Table 9-7. Pin State during Stop Grant Bus State

Signal	Type	State
A3-A2	O	Previous state
A31–A4	I/O	Previous state
D31–D0	I/O	Floated
BE3#–BE0#	O	Previous state
DP3–DP0	I/O	Floated
W/R#, D/C#, M/IO#	O	Previous state
ADS#	O	Inactive
LOCK#, PLOCK#	O	Inactive
BREQ	O	Previous state
HLDA	O	As per HOLD
BLAST#	O	Previous state
FERR#	O	Previous state
PCD, PWT	O	Previous state
PCHK#	O	Previous state
PWT, PCD	O	Previous state
SMIACT#	O	Previous state

9.6.3 CLOCK CONTROL STATE DIAGRAM

The following state descriptions and diagram show the state transitions during a Stop Clock cycle for the Military Intel486 processor. (Refer to Figure 9-9 for a Stop Clock state diagram.)

9.6.3.1 Normal State

This is the normal operating state of the processor.

9.6.3.2 Stop Grant State

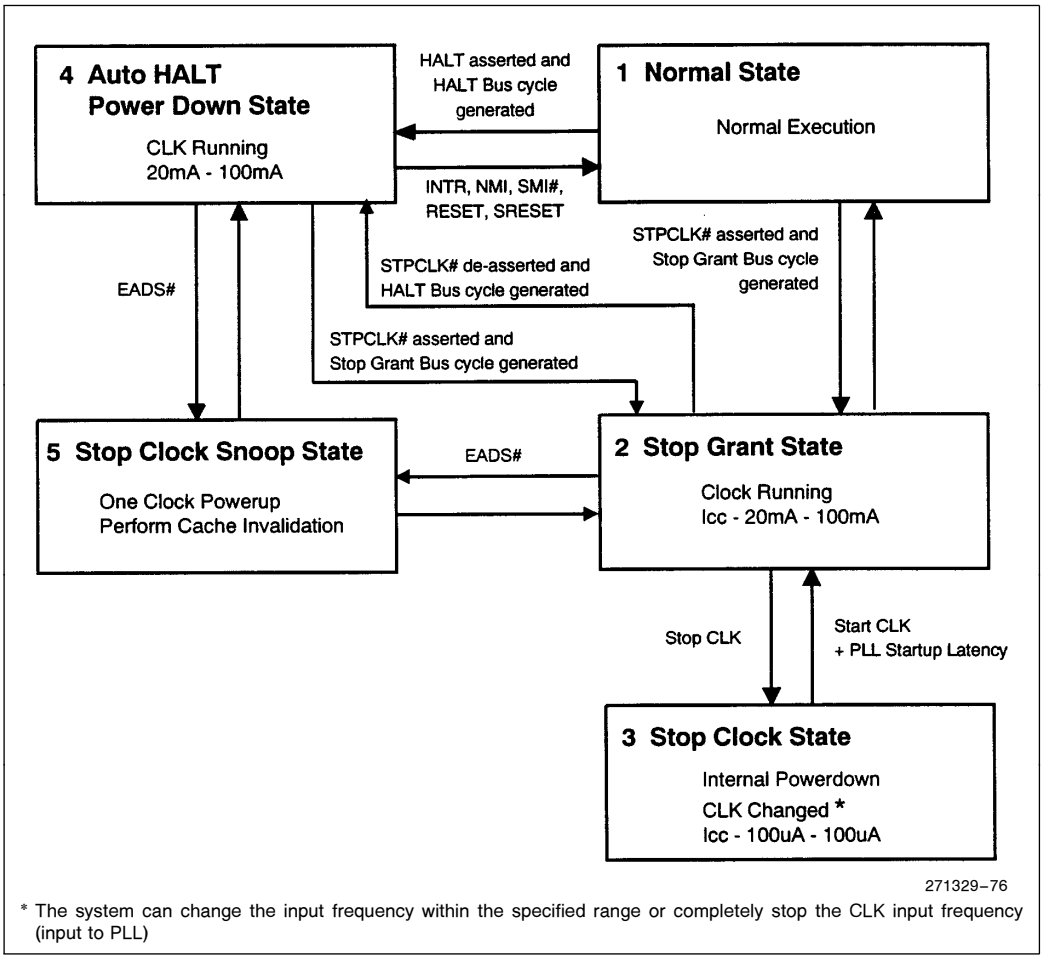
The **Stop Grant** state provides a fast wake-up state that can be entered by simply asserting the external STPCLK# interrupt pin. Once the Stop Grant bus cycle has been placed on the bus, and either RDY# or BRDY# is returned, the processor is in this state (depending on the CLK input frequency). The processor returns to the normal execution state 10–20 clock periods after STPCLK# has been de-asserted.

While in the Stop Grant state, the pull-up resistors on STPCLK#, CLKMUL (for the IntelDX4 processor) and UP# are disabled internally. The system must continue to drive these inputs to the state they were in immediately before the processor entered the Stop Grant state. For minimum processor power consumption, all other input pins should be driven to their inactive level while the processor is in the Stop Grant state.

A RESET or SRESET will bring the processor from the Stop Grant state to the Normal state. The processor will recognize the inputs required for cache invalidation's (HOLD, AHOLD, BOFF# and EADS#) as explained later in this section. The processor will not recognize any other inputs while in the Stop Grant state. Input signals to the processor will not be recognized until 1 CLK after STPCLK# is de-asserted (see Figure 9-10).

While in the Stop Grant state, the processor will not recognize transitions on the interrupt signals (SMI#, NMI, and INTR). Driving an active edge on either SMI# or NMI will not guarantee recognition and service of the interrupt request following exit from the Stop Grant state. However, if one of the interrupt signals (SMI#, NMI, or INTR) is driven active while the processor is in the Stop Grant state, and held active for at least one CLK after STPCLK# is de-asserted, the corresponding interrupt will be serviced. The Military Intel486 processor requires INTR to be held active until the processor issues an interrupt acknowledge cycle in order to guarantee recognition. (See Figure 9-10).

When the processor is in the Stop Grant state, the system is allowed to stop or change the CLK input. When the CLK input to the processor is stopped (or changed), the Military Intel486 processor requires the CLK input to be held at a constant frequency for a minimum of 1 ms before de-asserting STPCLK#. This 1-ms time period is necessary so that the PLL can stabilize, and it must be met before the processor will return to the Stop Grant state.



* The system can change the input frequency within the specified range or completely stop the CLK input frequency (input to PLL)

Figure 9-9. Military Intel486™ Processor Family Stop Clock State Machine

The processor will generate a Stop Grant bus cycle only when entering that state from the Normal or the Auto HALT Power Down state. When the processor enters the Stop Grant state from the Stop Clock state or the Stop Clock Snoop state, the processor will not generate a Stop Grant bus cycle.

9.6.3.3 Stop Clock State

Stop Clock state is entered from the Stop Grant state by stopping the CLK input (either logic high or logic low). None of the processor input signals should change state while the CLK input is stopped. Any transition on an input signal (with the exception of INTR, NMI and SMI#) before the processor has returned to the Stop Grant state will result in unpredictable behavior. If INTR is driven active while the CLK input is stopped, and held active until the processor issues an interrupt acknowledge bus cycle, it will be serviced in the normal manner. The system design must ensure the processor is in the correct state prior to asserting cache invalidation or interrupt signals to the processor.

The processor will return to the Stop Grant state after the CLK input has been running at a constant frequency for a period of time equal to the PLL start-up latency (see section 9.6.3.2). The CLK input can be restarted to any frequency between the minimum and maximum frequency listed in the AC timing specifications.

9.6.3.4 Auto HALT Power Down State

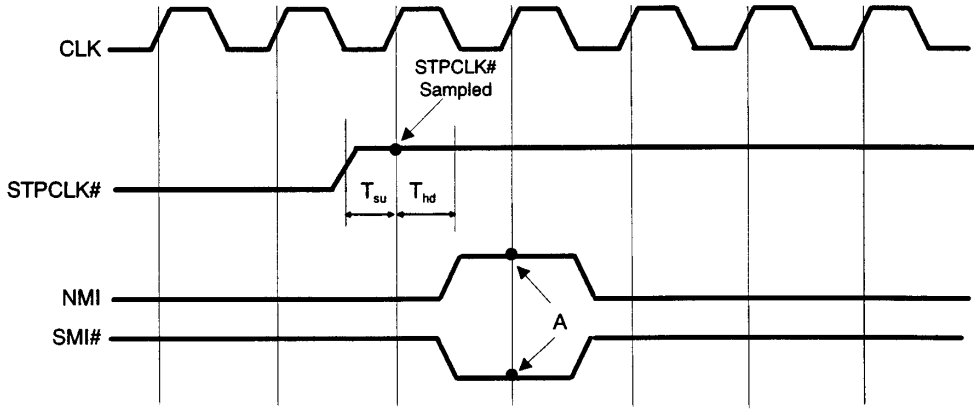
The execution of a HALT instruction will also cause the processor to automatically enter the **Auto HALT Power Down** state. The processor will issue a normal HALT bus cycle before entering this state. The processor will transition to the Normal state on the occurrence of INTR, NMI, SMI#, RESET, or SRESET.

The system can generate a STPCLK# while the processor is in the Auto HALT Power Down state. The processor will generate a Stop Grant bus cycle when it enters the Stop Grant state from the HALT state.

When the system de-asserts the STPCLK# interrupt, the processor will return execution to the HALT state. The processor will generate a new HALT bus cycle when it re-enters the HALT state from the Stop Grant state.

9.6.3.5 Stop Clock Snoop State (Cache Invalidations)

When the processor is in the Stop Grant state or the Auto HALT Power Down state, the processor will recognize HOLD, AHOLD, BOFF# and EADS# for cache invalidation. When the system asserts HOLD, AHOLD, or BOFF#, the processor will float the bus accordingly. When the system then asserts EADS#, the processor will transparently enter the **Stop Clock Snoop state** and will power up for 1 full core



A. Earliest time at which NMI or SMI# will be recognized.

Figure 9-10. Recognition of Inputs when Exiting Stop Grant State

clock in order to perform the required cache snoop cycle. It will then re-freeze the clock to the processor core and return to the previous state. The processor does not generate a bus cycle when it returns to the previous state.

A FLUSH# event during the Stop Grant state or the Auto HALT Power Down state will be latched and acted upon by asserting the internal FLUSH# signal for one clock upon re-entering the Normal state.

9.6.3.6 Auto Idle Power Down State

When the chip is known to be truly idle and waiting for a RDY# or BRDY# from a memory or I/O bus cycle read, the Military Intel486 processor will reduce its core clock rate to be equal to the external CLK frequency without affecting performance. When any RDY# or BRDY# is asserted, the part will return to clocking the core at the specified multiplier of the external CLK frequency. This functionality is transparent to software and external hardware.

9.6.4 STOP CLOCK SNOOP STATE (CACHE INVALIDATIONS)

When the processor is in the Stop Grant state or the Auto HALT Power Down state, the processor will recognize HOLD, AHOLD, BOFF#, and EADS# for cache invalidation. When the system asserts HOLD, AHOLD, or BOFF#, the processor will float the bus accordingly. When the system asserts EADS#, the processor will transparently enter the **Stop Clock Snoop state** and will power up in order to perform the required cache snoop cycle and write-back cycles. It will then refreeze the CLK to the processor core and return to the previous state (i.e., either the Stop Grant state or the Auto HALT Power Down state). The processor does not generate a bus cycle when it returns to the previous state.

9.6.5 SUPPLY CURRENT MODEL FOR STOP CLOCK MODES AND TRANSITIONS

Figures 9-11 and 9-12 illustrate the effect of different Stop Clock state transitions on the supply current of the Military Intel486 processor.

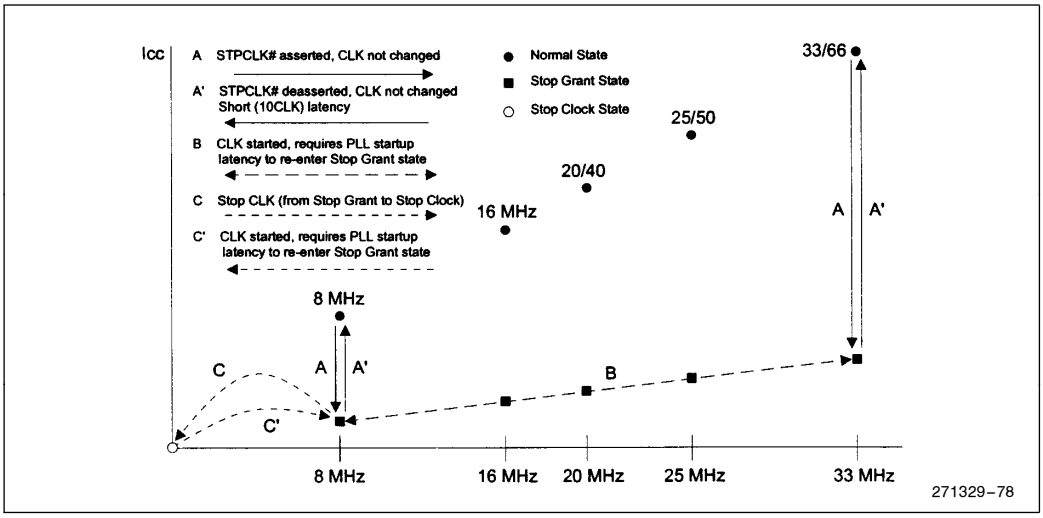


Figure 9-11. Supply Current Model for Stop Clock Modes and Transitions for the Military Intel486™ Processor

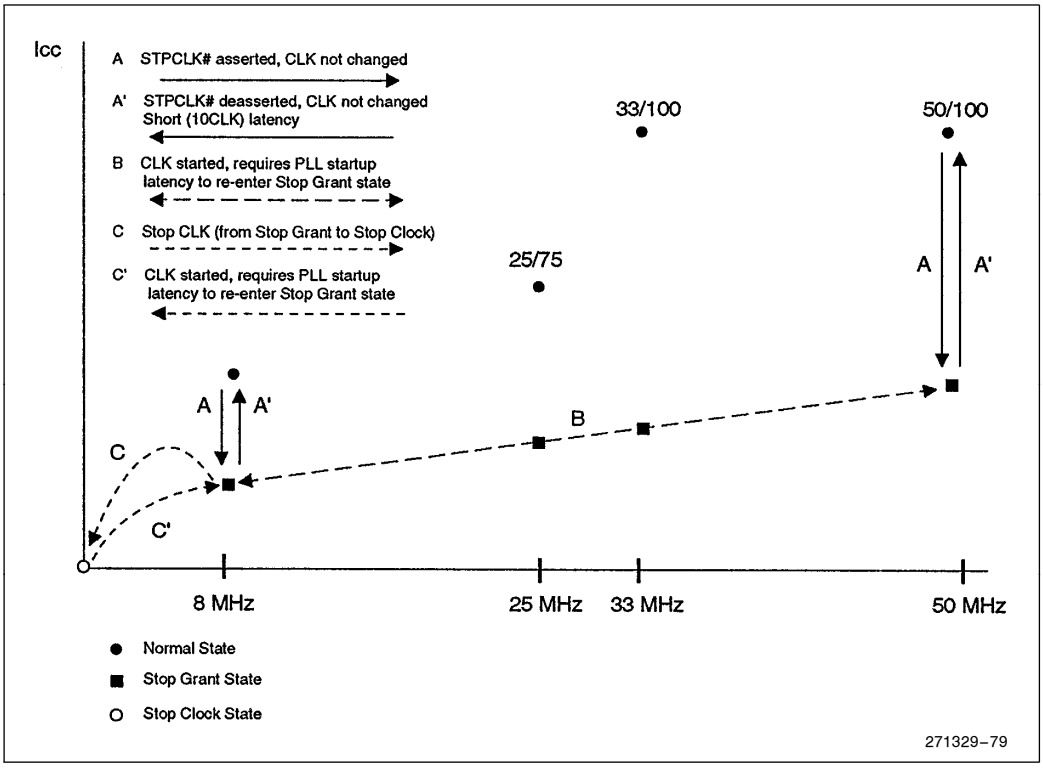


Figure 9-12. Supply Current Model for Stop Clock Modes and Transitions for the IntelDX4™ Processor